2015-03

Improving sector hash carving with rule-based and entropy-based non-probative block filters

Gutierrez-Villarreal, Francisco Javier

Monterey, California: Naval Postgraduate School

http://hdl.handle.net/10945/45194

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

IMPROVING SECTOR HASH CARVING WITH RULE-BASED AND ENTROPY-BASED NON-PROBATIVE BLOCK FILTERS

by

Francisco Javier Gutierrez-Villarreal

March 2015

Thesis Advisor:	Michael R. McCarrin
Thesis Co-Advisor:	Joel D. Young

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave Blank)*	**2. REPORT DATE** 03-27-2015	**3. REPORT TYPE AND DATES COVERED** Master's Thesis 01-07-2013 to 03-27-2015

4. TITLE AND SUBTITLE

IMPROVING SECTOR HASH CARVING WITH RULE-BASED AND ENTROPY-BASED NON-PROBATIVE BLOCK FILTERS

5. FUNDING NUMBERS

6. AUTHOR(S)

Francisco Javier Gutierrez-Villarreal

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: NPS.2011.0024-CR01-EPS-A

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Digital forensic investigators have traditionally used file hashes to identify known content on searched media. Recently, sector hashing has been proposed as an alternative identification method, in which files are broken up into blocks, which are then compared to sectors on searched media. Since sectors are read sequentially without accessing the file system, sector hashing can be parallelized easily and is faster than traditional methods. In addition, sector hashing can identify partial files, and does not require an exact file match. In some cases, the presence of even a single block is sufficient to demonstrate with high probability that a file resides on a drive. However, non-probative blocks, common across many files, generate false positive matches; a problem that must be addressed before sector hashing can be adopted. We conduct 7 experiments in two phases to filter non-probative blocks. Our first phase uses rule-based and entropy-based non-probative block filters to improve matching against all file types. In the second phase, we restrict the problem to JPEG files. We find that for general hash-based carving, a rule-based approach outperforms a simple entropy threshold. When searching for JPEGs, we find that an entropy threshold of 10.9 gives a precision of 80% and an accuracy of 99%.

14. SUBJECT TERMS

Digital Forensics, Digital Fingerprinting, Approximate Matching, Sector Hashing, Distinct Blocks, Probative Blocks, Block Filtering, Hash Databases, Hash Carving

15. NUMBER OF PAGES 91

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

IMPROVING SECTOR HASH CARVING WITH RULE-BASED AND ENTROPY-BASED NON-PROBATIVE BLOCK FILTERS

Francisco Javier Gutierrez-Villarreal
Civilian, Department of the Navy
B.S., Santa Clara University, 2011

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2015

Author:	Francisco Javier Gutierrez-Villarreal
Approved by:	Michael R. McCarrin Thesis Advisor
	Joel D. Young Thesis Co-Advisor
	Peter Denning Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Digital forensic investigators have traditionally used file hashes to identify known content on searched media. Recently, sector hashing has been proposed as an alternative identification method, in which files are broken up into blocks, which are then compared to sectors on searched media. Since sectors are read sequentially without accessing the file system, sector hashing can be parallelized easily and is faster than traditional methods. In addition, sector hashing can identify partial files, and does not require an exact file match. In some cases, the presence of even a single block is sufficient to demonstrate with high probability that a file resides on a drive. However, non-probative blocks, common across many files, generate false positive matches; a problem that must be addressed before sector hashing can be adopted. We conduct 7 experiments in two phases to filter non-probative blocks. Our first phase uses rule-based and entropy-based non-probative block filters to improve matching against all file types. In the second phase, we restrict the problem to JPEG files. We find that for general hash-based carving, a rule-based approach outperforms a simple entropy threshold. When searching for JPEGs, we find that an entropy threshold of 10.9 gives a precision of 80% and an accuracy of 99%.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

I would like to thank my thesis advisor, Michael McCarrin, for his gentle guidance throughout the thesis process. I am deeply grateful for your unwavering commitment to my success and for always being available to give a helping hand.

I would also like to thank Dr. Joel Young for introducing me to the field of computer forensics and for being my co-advisor. Thank you for all of your guidance and feedback.

Thank you Joe Welch and Alison Kerr for always believing in me and for helping me navigate through all of the opportunities available at NPS since I first arrived as a Hartnell STEM intern. Thank you Dr. David Alderson for being an outstanding mentor and for allowing me to learn what computer programming was all about.

Thanks also to Dr. Cynthia Irvine, the Scholarship-For-Service program, and all of my NPS professors. I am eternally grateful for the opportunity to have attended such a prestigious institution.

Thank you Katherine, Michael, and Austin. You guys were great classmates and even better friends.

I would also like to thank my family for all of the help they provided throughout these past two years. I would not have been able to complete this program without your love and support.

Finally, I would like to thank God for all of His blessings and for guiding me through the challenges of life.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: Introduction

Digital forensic investigators are overwhelmed by the volume of storage media and the proliferation of storage formats they encounter in the course of their work. Technological advances have led to an increase in storage media capacity and decrease in costs. Today, the typical consumer owns multiple hard drives with storage capacities in the terabyte range. These drives take hours to read and image, and much longer to analyze. It is also common for consumers to own a wide variety of devices, each running different operating systems with different, often proprietary, storage formats. In addition, the adoption of cloud computing, has led to the synchronization of copies of consumer data across multiple machines, including unconventional devices such as video game consoles. These developments pose challenges to investigators attempting to identify content of interest on digital storage media.

1.1 Traditional Hash-Based File Identification

Investigators are often interested in identifying files with known content, such as contraband images or video files, on seized storage media. These files are referred to as target files.

Forensic investigators have traditionally used cryptographic hash functions to create digital signatures of target files they need to identify. These hashes are used to scan digital storage media for matches. If any matches are returned, there is a strong probability that the file in question resides on that drive. Confidence in the matches stems from the fact that cryptographic hash functions are collision resistant, which means that it is highly unlikely that a cryptographic hash function will produce the same output for two distinct files [1].

Comparing hashes instead of files significantly reduces the workload of a digital forensic analyst. The hash matching process is automated so that analysts no longer have to inspect a file's actual content. While other automated comparisons could be performed, such as byte-by-byte comparisons, hash-based comparisons are preferred, since hashes are easier to store and are faster to compare. However, due to the avalanche effect (Section 2.2), it is possible that many near matches will not be found. This is a problem because it is trivial for criminals to try to hide their tracks by making a small modification to a file, completely changing its hash value.

1.2 Sector Hashing

An alternative approach for identifying target files on digital storage media involves looking at the hashes of chunks of files instead of the hash value of an entire file. Garfinkel et al. [2] propose an approach where target files are broken up into chunks of 4096 bytes in order to match the sector size used on most drives. These chunks of files are known as blocks, and their hashes as block hashes. When a hard drive is searched for target files, 4096-byte sectors are read, hashed, and compared to block hashes of target files.

One of the benefits of sector hashing is that it is relatively file system agnostic. A drive is processed by reading sectors sequentially off of storage media rather than by seeking back and forth following file system pointers. This is an improvement over traditional methods, since it allows the drive to be read sequentially, which minimizes seek time, and is easily parallelized by splitting the image into segments and processing them simultaneously. Another benefit of sector hashing is that files that have been deleted or partially overwritten can also be found, since the entire drive is analyzed, not just allocated space. Traditional hashing methods require that an entire file be present on a drive for a match to be returned, but sector hashing is not subject to this limitation. The investigator may determine that as little as one matching block is enough evidence to conclude that a file resides on a drive. Finally, statistical sector sampling can be used to quickly triage a drive and determine the presence of target files [2], [3].

1.3 Probative Blocks

File identification using sector hashing relies on the ability to correlate the presence of a sector with the presence of a file. Garfinkel and McCarrin [4] define a probative block as a block that can be used to demonstrate with high probability that the file in question resides on a drive. If a copy of a probative block is found on a storage device, then there is strong evidence that the file is or was once present on that device. An example of how probative blocks can be used to identify target content is discussed below.

1.3.1 Probative Block Scenario

Suppose that a terrorist network is circulating a video file that is unique to their organization. Forensic investigators learn of the video after recovering it from a seized hard drive belonging to one of the terrorists. The investigators take block hashes from the video file and add them to a database.

A group of suspected terrorists are later captured. External hard drives, mobile phones, and computers are seized. One of the men arrested had once possessed a copy of the video file on his laptop, but had deleted it long ago. The file was no longer accessible through the file system, and most of it had been overwritten by new data.

In order to detect the presence of the incriminating file, forensic investigators begin their triage process using sector hashing. They compare the sector hashes on the seized drives to the block hashes in their database of target content. One of the drives returns multiple matches for probative blocks in their database, blocks that have only been observed as part of the terrorist network's video file. This information leads the forensic investigators to believe that the video file was once present on the drive and that the owner of that drive is associated with the terrorist network.

1.3.2 Which Blocks are Probative?

The above scenario was an ideal example of using probative blocks to find target files on digital storage media. Since probative blocks are so useful, we need to know which blocks in a target file can be considered probative blocks that can later be used for file identification. We also need to know which blocks are common across many files. For example, many files share similar file headers or contain the same data structures, such as color maps in image files. Matches to these common blocks are false positives for a target file, since they do not match content generated by a user and can be found across multiple distinct files.

However, classifying blocks as probative is no easy task, a problem that must be addressed if sector hash matching is to be adopted by forensic investigators. It is very difficult to know if a block is truly probative or not. A block that may appear to be probative in one data set may be shown to be common to many files once a larger data set is considered [4].

Instead of trying to find probative blocks, it may be easier to try to find non-probative blocks and eliminate them from consideration. Many non-probative blocks contain patterns that are easy to identify, such as repeating n-grams, or contain low-entropy data, such as a block of all NULLs. Unfortunately, not all common blocks fit these descriptions. Some common blocks arise from programmatically generated processes in different software programs. These blocks are difficult to catch, since they do not follow any obvious pattern and resemble data generated by users.

1.4 Thesis Contributions

In order to eliminate common block matches from consideration, Garfinkel and McCarrin [4] devised a set of 3 *ad hoc* rules, which they refer to as the histogram, ramp, and whitespace rules. The *ad hoc* rules attempt to filter out blocks that are not probative—that is, blocks that do not help the forensic investigator demonstrate that a target file resides on a drive.

This thesis evaluates Garfinkel and McCarrin's rules by using the Govdocs corpus as our set of target files and 1,530 drives from the Real Data Corpus (RDC) as our set of searched media. We compare the performance of the *ad hoc* rules to simple entropy thresholds and find that an entropy threshold of six is roughly equivalent to the three *ad hoc rules*. In addition, we find that the three ad hoc rules are redundant and can be replaced by a single rule. Finally, we focus our search efforts on JPEG files and find that classifying blocks with an entropy value below 10.9 as non-probative allows us to identify Govdocs JPEG files that reside on the RDC with 80% precision and 99% accuracy.

The remainder of this thesis is organized as follows. Chapter 2 provides a technical background. Chapter 3 discusses previous work related to alternatives to file hash matching. Chapter 4 describes our methodology for building a block hash database out of the Govdocs corpus, describes our method for scanning RDC drives for matches, and describes our sector hashing experiments. Chapter 5 describes our results. Finally, in Chapter 6, we discuss our conclusions and future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Technical Background

This chapter provides a background on cryptographic hash functions and their application to identifying known content on searched media. We discuss traditional file hash matching techniques and their limitations. In addition, we provide a background on sector hashing and discuss the benefits and limitations of sector hash matching.

2.1 Cryptographic Hash Functions

A hash function is an algorithm that maps an input of arbitrary length to an output of fixed length. The output is referred to as a hash, or digest. For example, the Message Digest 5 (MD5) hash function takes an input of arbitrary length and creates a 128-bit output [5] (see Figure 2.1). Hash functions are deterministic, which means that the output of a hash function will not change if the same input is used.

A cryptographic hash function is a hash function with the following additional properties [1]:

1. Pre-image resistance: Given the output of a hash function, it is computationally infeasible to find an input that hashes to that output.
2. Second pre-image resistance: Given an input, it is computationally infeasible to find another distinct input that will hash to the same value as the first input.
3. Collision resistance: It is computationally infeasible to find two different inputs that will hash to the same value.
4. Avalanche effect: A small change in the input of a hash function will result in a very large change to the output.

These properties make cryptographic hash functions useful for a wide variety of applica-

```
Input: hello world!

md5: c897d1410af8f2c74fba11b1db511e9e
```

Figure 2.1: Message digest 5 (MD5) output

tions. In particular, the property of collision resistance, combined with the fact that the functions are deterministic, allows digital forensic investigators to treat the hash of an input as a fingerprint for that input, since it is very unlikely that another distinct input will produce the same hash.

For example, MD5, one of the most popular cryptographic hash functions used by digital forensic practitioners today, produces a 128-bit output, which means that there are 2^{128} different possible hash values [6]. The number of inputs required for a 50% probability of an MD5 collision to occur by chance is 2^{64}, or $1.8x10^{19}$ [6].

2.1.1 Applications of Cryptographic Hash Functions

For this thesis, we focus on using cryptographic hashes for content identification. Since it is extremely unlikely that two non-identical digital artifacts will hash to the same value due to the collision resistance property, digital forensic practitioners use cryptographic hash functions to create digital fingerprints of files. These fingerprints, or hashes, can then be added to whitelists and blacklists to assist investigators in identifying targeted content on digital media.

2.2 Whitelists and Blacklists

Whitelists and blacklists significantly reduce a forensic investigator's workload. Whitelists help achieve data reduction by eliminating spurious matches and noise, while blacklists help investigators focus on suspicious data.

2.2.1 Whitelists

In digital forensics, whitelisting is the process of taking known good or trusted files and adding them to a to-be-ignored database [7]. Typically, only the hashes of whitelisted files, rather than their full content, are added to the database; this greatly reduces storage requirements and increases speed. When whitelisted hashes are encountered again, an investigator can simply ignore them, since it has been determined that the content they represent does not warrant further investigation.

An example of a whitelist database used by digital forensic investigators is the NIST National Software Reference Library Real Data Set (NSRL RDS) [8]. The NSRL RDS contains collections of whitelisted hashes, obtained from software files that originate from known sources, such as commercial vendors (e.g., Microsoft, Adobe). The NSRL RDS helps forensic investigators identify files from known applications, which reduces the workload involved in identifying files that may be important for criminal investigations [8].

2.2.2 Blacklists

A blacklist is a list of targeted files that are known to be malicious, unauthorized, or illegal [7]. Files with hashes that match to a blacklist can be tagged as suspicious and investigated further.

An example of a blacklist used by digital forensic investigators is a database held by the FBI's Child Victim Identification Program (CVIP) [9]. This database contains hashes of images that depict Sexual Exploitation of Children (SEOC) offenses. Analysts can compare the hash values of files in the CVIP database to hash values from seized files to quickly identify if a SEOC image is present on searched media.

2.2.3 Limitations of Whitelisting and Blacklisting

While whitelists and blacklists help forensic investigators find previously known content, they do not help identify material that an investigator is not looking for in advance. For example, the CVIP database will not help identify new cases of child exploitation, since it can only find images that match to a database of victims that have been previously identified. In addition, storage requirements may be an issue. For example, if one is attempting to store a whitelist of all files known to be benign, it becomes difficult to store and search through these files.

2.3 File Hash Matching

Forensic investigators have traditionally used file hashing techniques along with hash databases to determine if targeted content is contained within seized storage media, such as hard drives or mobile phones [10]. By comparing the hash values of files found on seized media to a database of known file hashes, investigators can identify the presence of illegal, malicious, or unauthorized files. File hash identification greatly reduces the amount of time it takes an investigator to search a storage device for targeted content, since an investigator no longer has to manually examine the content of each file. However, the effectiveness of file hash matching is limited by several constraints. First, file hash matching relies on exact matches. If a target file on searched media is modified, a match will not be returned. Also, file hash matching relies on the file system to provide access to target files, often requiring a recursive walk of the entire directory structure. This process may take a considerable amount of time, and does not work on unallocated space, unless the exact byte offset of the file is known. Although deleted files can sometimes be recovered through "carving," or reassembling blocks in unallocated space, reassembly algorithms are often computationally expensive and recovery is not guaranteed.

2.3.1 Avalanche Effect

Cryptographic hash functions are designed such that a small change to the input results in a very large change to the output [1]. This property is known as the avalanche effect. Even a single character change results in a completely different hash value, as seen in Figure 2.2. While this property provides the cryptographic hash function with pre-image resistance, it also poses a problem for digital forensic investigators who are using hash databases that rely on exact matches.

Suppose that a forensic analyst is searching a seized hard drive for the presence of known incriminating files. During the course of the investigation, the analyst creates a copy of the seized drive and hashes all of the files. The analyst then looks for hash matches to their database of known target files. However, if the target files on the seized drive have been corrupted, partially deleted, or modified, they will not match the hashes in the database. Valuable evidence for an investigation can be missed this way. Thus, it is necessary to find a more robust method for identifying targeted content.

2.4 Sector Hash Matching

An alternative approach for identifying target files on digital storage media involves matching chunks of files against chunks of the same size on storage media. We focus primarily on this approach. We adopt the terminology defined by Taguchi [3] where a block is defined as a fixed-length segment of a file and a file block as a 4096-byte chunk of file. Sectors are defined as 4096-byte sections of the searched disk.

As with file hashes, block hashes from target files can be stored in a database for future reference. When a hard drive is searched, 4096-byte sectors are read. These sectors are

```
Input: The admin password is secret
md5: 9fae77b095e4a834b48534282a34bc16

Input: The admin password is Secret
md5: 6c2a077b7c4b5004466039b26f464c85
```

Figure 2.2: An illustration of the avalanche effect: a change in a single character results in a completely different hash value.

then hashed and compared to block hashes in the database.

Traditional file hash identification methods require that an entire unmodified target file be present on a drive for a match to be returned (i.e., an exact match). However, sector hashing can detect files that have been modified, deleted, or partially overwritten. If enough blocks from a target file are found on a seized drive, the digital forensic investigator may determine that there is enough evidence to conclude that the file resides on that drive.

There are limitations to sector hashing, however. One limitation is that generating block hash databases requires that more hashes be stored and queried than traditional file hash databases. In addition, interpreting matches to block hashes is more complex. Not all blocks are unique to the file that they belong to and may be found across multiple files. Garfinkel and McCarrin define three types of blocks [2], [4].

2.4.1 Types of Blocks

Garfinkel defines a *distinct block* as a block that will not arise by chance more than once. If a copy of a distinct block is found on a storage device, then there is strong evidence that the file is or was once present on that device. If a block is shown to be distinct throughout a large and representative corpus, then the block can be treated as if it was universally distinct [2].

A *common block* is a block that appears in more than one file. These blocks are generally of limited value to forensic investigators, and frequently indicate data structures common across files [4].

A *probative block* is a block that can be used to state with high probability that a file is or once was present on a drive [4].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3: Related Work

The need to analyze ever greater quantities of data has motivated the development of alternatives to traditional forensic methods. This chapter will discuss recent innovations in hash matching techniques as well as the applications of these techniques in identifying targeted content on storage media.

3.1 Challenges to Traditional File Identification

During the “Golden Age” of digital forensics (1999-2007) forensic investigators were able to use digital forensics to trace through a criminal’s steps and recover incriminating evidence [11]. Garfinkel states that forensic investigators were aided by the following circumstances [11]:

- The widespread prevalence of Microsoft Windows XP.
- The number of file formats of forensic interest were relatively small, and consisted mostly of Microsoft Office documents, JPEGs, and WMV files.
- Investigations predominantly involved a single computer system that had a storage device with standard interfaces

Garfinkel [11] predicts that current forensics tools and procedures will fail to keep up with the rapid pace of market-driven technological change. Consumers now have access to affordable terabyte storage drives, cloud computing, and personal data encryption. It is also common for consumers to own a wide variety of devices (e.g., cell phones, tablets, gaming consoles), each running different operating systems with different storage formats. Garfinkel et al. [2] explain that many file hash identification techniques involve reading an entire file system, which takes a significant amount of time. The tree structure of modern file systems is also an obstacle, as it makes it difficult to parallelize disk analysis. This new age of digital forensics leaves investigators with more data than can be processed. To make matters worse, the growing diversity of operating systems, applications, and storage formats makes it likely that investigators will miss information pertinent to their investigations.

3.2 File Hash Matching Limitations

Young et al. [12] argue that file hashes are not a robust method of identification. First, using file hashes to identify known content is limited by the *avalanche effect*. This allows individuals with incriminating files to avoid detection by making trivial changes to their files, such as by appending a few random bytes. Second, if a target file has been deleted and partially overwritten by new data, the hash values of the file fragments will likely no longer match the original. A third limitation of file hashing is that if sections of a file are damaged or corrupted, file hash identification will fail once again. Since a file hash database contains the hashes of complete, unmodified target files, file hash identification will fail to detect the presence of target files with any alterations—no matter how small.

Furthermore, recent technological advances threaten to severely reduce the usefulness of file hash identification. File hash identification techniques do not scale well to the rapid increase in the size and type of data seized during investigations [11] .

3.3 Alternatives to File Hash Matching Techniques

In order to overcome some of the limitations imposed by file hash matching techniques, several new techniques have been proposed.

3.3.1 Context Triggered Piecewise Hashing

Kornblum developed a technique to identify similar files using Context Triggered Piecewise Hashes (CTPH) [13]. Similarity matching helps forensic investigators overcome the exact match problem caused by the avalanche effect. CTPH can be used to identify homologous sequences between a target file and modified versions of that file. Files with homologous sequences have large sequences of identical bits in the same order. To determine a similarity score between two files, Kornblum takes each file and generates a string of hashes that is up to 80 bytes long and that consists of smaller, 6-bit piecewise hashes. Similarity between two hashes, h1 and h2, is calculated using a weighted edit distance score to measure how many changes, deletions, or insertions it would take to mutate h1 into h2. Files are then assigned a final score between 0 and 100, with 0 representing no homology and 100 representing almost identical files.

Kornblum implemented the CTPH technique in the *ssdeep* program [14]. Using *ssdeep*, Kornblum was able to match a file containing Abraham Lincoln's Gettysburg Address with a modified version that included several insertions and deletions. *ssdeep* gave the modified version a score of 57, indicating significant similarity between the two files.

The CTPH technique also works for finding partial files. Kornblum took a file of size 313,478 bytes and split it into two smaller files. File 1 consisted of the first 120,000 bytes from the original file while File 2 consisted of the last 113,478 bytes. After running both files through *ssdeep*, File 1 received a score of 54 while File 2 received a score of 65, indicating that both files had high similarities to the original file [13].

3.3.2 Similarity Digests

Roussev [10] also developed a technique for identifying similar files. His technique involves identifying statistically-improbable 64-byte features. Statistically-improbable features are features that are unlikely to occur in other data objects by chance. Roussev implemented his technique for feature selection in *sdhash* [15].

sdhash views a file as a series of overlapping 64-byte strings called features. All possible 64-byte sequences are considered for each file using a sliding window. Features are then given a popularity rank and are selected as characteristic features if their rank exceeds a threshold.

The *sdhash* feature selection algorithm attempts to avoid choosing weak features that are likely to occur in multiple data objects by chance. Roussev finds that features that have low entropy tend to occur in multiple files and are therefore not statistically-improbable [10]. Features with near maximum entropy are also disqualified since, Roussev argues that these features likely arise from algorithmically generated content, such as Huffman and Quantization headers in JPEGs, which have high entropy, but are likely to occur across multiple files. By reducing the number of weak features, *sdash* can reduce the rate of false positive matches.

After characteristic features have been selected, the features are hashed using SHA-1, which creates a 160-bit digest. This digest is then split into 32-bit subhashes. Each subhash is inserted into a bloom filter with a maximum size of 160 or 192 elements, depending on whether sdhash is in continuous or block mode. When a bloom filter reaches its maximum size, a new one is created. A similarity digest (SD) for an object is created by concatenating all its bloom filters.

Two similarity digests are compared by taking the smaller SD and comparing each of its bloom filters with each of the bloom filters in the larger SD. For each filter in the smaller SD, the maximum similarity score is selected. A final score is calculated from the average of the maximum scores.

sdhash fixes the exact match problem but requires $O(nm)$ comparison time, where n and m are the total number of bloom filters in target and reference sets. While *sdhash* may work for small datasets, it is not scalable.

3.3.3 Sector Hash Matching

In order to address some of the limitations of the methods described above, Young et al. [12] propose a different framework for analyzing data on a disk. Instead of analyzing a drive through file-level hashes and relying on the file system, the authors propose hashing small blocks of bytes in order to identify targeted content through a procedure known as sector hashing. Their sector hashing approach takes blocks of either 512 bytes or 4096 bytes in order to align with the sectors on most drives. These hashes can then be compared against pre-built block hash databases for matches. Unlike bloom filters, block hashes can easily be organized in a tree structure for scalable, $O(logN)$ lookup times. Garfinkel [2] found that block sizes of 4096 work best on most file systems and take up 1/8 less storage space than block sizes of 512 bytes.

However, sector hashing might not work for matching files that are not sector aligned, especially if the underlying drive image uses 512-byte sectors, and the file system begins at an offset that is not an even multiple of 4096. When target blocks can start on any sector boundary, only target blocks that are block aligned can be found, since non-block aligned target blocks will only contain part of the data being searched for. In his thesis, *Optimal sector sampling for drive triage* [3], Taguchi performs sector sampling experiments to find the presence of target data on a drive by using a target size of 4096 bytes. He defines the transaction size as the amount of data read from searched media. Taguchi states that transaction sizes greatly affect the probability of locating target data on a drive.

In order to deal with the alignment problem, Taguchi [3] proposes using transaction sizes that are larger than target blocks. Larger transaction sizes can be used to find non-block aligned blocks as long as the transaction fully covers the block. When sector sampling using 4096-byte (4KiB) target blocks, Taguchi proposes using a transaction size that is at a minimum double the target block size, since the sample read will contain enough block offsets so that at least one will be aligned on a 4KiB boundary. In his experiments, Taguchi found that a 65,536-byte (64KiB) transaction size was the ideal transaction size when conducting 90% target confidence sampling on a terabyte drive.

3.3.4 Benefits of Sector Hashing

There are many advantages to sector hashing. First, since sectors are being read as bytes directly off of storage media, there is no longer a need to go through a file system. This allows for drive images to be split up into chunks and analyzed in parallel, greatly increasing search speeds. In addition, reading bytes directly off a disk provides a file-system-agnostic way to search for content. This is significant considering the rapid growth in the different types of storage formats available today. Another advantage of sector hashing is that it helps deal with the avalanche effect problem. By splitting a file into multiple blocks, the probability of finding a match increases, since we are no longer searching for a single file hash.

An added benefit to sector hashing is that statistical sampling can be used to to detect the presence of targeted content on a drive. This procedure can be described by the Urn Problem of sampling without replacement [3].

$$P = \prod_{i=1}^{n} \frac{((N-(i-1))-M)}{(N-(i-1))}$$

N is the number of sectors on the digital media

M is the size of the target data as a multiple of the sector size

n is the number of sectors sampled

Garfinkel et al. [2] used random sampling to scan a terabyte drive looking for 100MB of target content in 2–5 minutes, as opposed to hours using traditional file forensics. By sampling 50,000 sectors from the terabyte drive, there is a 99% chance that at least one sector of the 100MB file will be found. Taguchi [3] also used random sampling to determine that less than 10MiB of target data was present on a 500GB drive with a 90% confidence by sampling the drive for only 15 minutes.

3.4 Distinct Blocks

Young et al. [12] examined three large file corpora (GovDocs, OpenMalware 2012, 2009 NSRL RDS) and found that the vast majority of both executable files and user-generated content were distinct. The authors hashed these several million file corpora into 512-byte and 4096-byte blocks and classified the blocks according to the number of times that they appeared. Blocks that occurred only once were termed singleton blocks. Blocks that occurred only twice were termed paired blocks. Finally, blocks that occurred more than twice were termed common blocks.

Table 3.1 shows their results [12]. Clearly, singleton blocks comprised the vast majority of the total blocks found in the file corpora. These blocks were considered distinct, and the authors proposed that they could be used to identify targeted content, since they were not likely to occur by chance anywhere other than in copies of the files they belonged to.

	Govdocs1		**OCMalware**		**NSRL2009**	
Total Unique Files	974,741		2,998,898		12,236,979	
Average File Size	506 KB		427 KB		240 KB	
Block Size: 512 Bytes						
Singletons	911.4M	(98.93%)	1,063.1M	(88.69%)	N/A	N/A
Pairs	7.1M	(.77%)	75.5M	(6.30%)	N/A	N/A
Common	2.7M	(.29%)	60.0M	(5.01%)	N/A	N/A
Block Size: 4096 Bytes						
Singletons	117.2M	(99.46%)	143.8M	(89.51%)	567.0M	(96.00%)
Pairs	0.5M	(.44%)	9.3M	(5.79%)	16.4M	(2.79%)
Common	0.1M	(.11%)	7.6M	(4.71%)	7.1M	(1.21%)

Table 3.1: Singleton, pair, and common blocks in the Govdocs, OCMalware, and NSRL2009 file corpora. Source: Young et al. [12]

Foster [16] found that most nondistinct blocks were low entropy or contained repeating byte patterns. She also found multiple occurrences of common blocks that were made up of data structures present in Adobe files or Microsoft Office files. These findings show that sector hashing presents its own set of problems. Since sector hash matches are based entirely on byte-level analysis, it is possible that there will be matches for file blocks that may have little content-level similarity. For example, Microsoft Office Documents that have been created using the same template will share many blocks in common [16] . These matches are false positives that distract investigators from finding their intended targets. Investigators searching for target files do not care about the "envelope" containing the file content—they are interested in the content itself.

3.5 Probative and Common Blocks

Garfinkel and McCarrin [4] propose using hash-based carving to search for target files on a storage device. Their approach involves comparing hashes of 4096-byte blocks on the storage device to same sized blocks in a database of target file blocks. Their technique expands on the work done by Foster [16] and Young et al. [12] by using block hashing to find probative blocks on searched drives. They make a distinction between probative blocks and distinct blocks due to the fact that it is difficult to know when a block is truly distinct. Contrary to what Garfinkel [2] hypothesized, Garfinkel and McCarrin [4] found that a block that appears to be rare in a large data set cannot be assumed to be universally distinct just because it only appears once in a large data set. They found that many of the singleton blocks identified by Foster [16] were rare enough to only appear once in the million file Govdocs corpus, but common enough to be found among other files when a larger file corpus was examined. Matches on these blocks are false positives for a target file, since they do not match content generated by a user, but instead match data structures common across files.

In order to filter out blocks that appear in multiple files, Garfinkel and McCarrin devised three rules [4]:

1. The Ramp rule attempts to filter out blocks that contain arrays of monotonically increasing 16-bit numbers padded to 32-bits. These blocks are common in Microsoft Office Sector Allocation Tables (SAT). If half of the 4096-byte block contains this

pattern, the block is eliminated from consideration as a probative block.

2. The Histogram rule attempts to filter out blocks with repeated 4-byte values, which were found to be common among Apple QuickTime and Microsoft Office file formats. To apply this rule, a block is read as a sequence of 1024 4-byte integers and a histogram is computed of how many times 4-byte values appear. If a value appears more than 256 times, the rule is triggered and the block is filtered. If there are only one or two distinct 4-byte values, the rule is also triggered.

3. The Whitespace rule attempts to filter out blocks that contain over three-quarters of whitespace characters, such as spaces, tabs, and newlines. These blocks were found to be common among JPEGs produced with Adobe PhotoShop.

Garfinkel and McCarrin [4] also considered using an entropy threshold to filter out common blocks. Foster [16] states that most blocks with low entropy are common, and that blocks with high entropy are unlikely to be present in two distinct files [16]. Garfinkel and McCarrin found that blocks flagged by their *ad hoc* rules support Foster's statement that common blocks have low entropy. They found that [4]:

1. Blocks flagged by the ramp test had an entropy score around 6.
2. Blocks flagged by the whitespace test had an entropy score between 0 and 1.
3. Most blocks flagged by the histogram test had an entropy score of 5 or less (although the entropy score for these blocks ranged from 0 to 8.373)

Garfinkel and McCarrin [4] determined that an entropy threshold of 7 identified an equal number of non-probative blocks as their *ad hoc* rules. However, they also found that the entropy method suppressed a far greater number of probative blocks than the *ad hoc* rules. For 12.5% of their target files, the entropy method flagged over 90% of those files' blocks, while only 1.6% of target files had over 90% of their blocks flagged with the *ad hoc* rules. Since the probability of reconstructing a file is greatly reduced when fewer than 10% of the file's blocks remain, Garfinkel and McCarrin state that they preferred the *ad hoc* rules to the entropy method for their use case. However, they also caution that this conclusion may not hold for the general case.

This thesis expands on Garfinkel and McCarrin's [4] work by examining their methods against a much larger data set, the Real Data Corpus. Our methodology is explained in Chapter 4 and our results in Chapter 5.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Methodology

We begin this chapter by discussing the data sets and tools used in our experiments. We then provide an overview of our experimental process and provide an outline for each experiment. In Sections 4.4, 4.5, and 4.6, we describe building a sector hash database and discuss using that database to scan drive images for matches, as well as our process for interpreting matches. We conclude the chapter by explaining our iterative process for experiment design, which led to the development of our second phase of experiments.

4.1 Datasets

To conduct our sector hashing experiment, we need a dataset to represent our target files and another dataset to represent our searched media. For our set of target files, we use the Govdocs corpus. For our set of searched media, we use drives from the Real Data Corpus.

4.1.1 Govdocs Corpus

The Govdocs corpus is a collection of nearly 1 million files consisting of a diverse set of formats (e.g., PDF, JPG, and PPT files) taken from publicly available U.S. government websites in the .gov domain [17]. We use all of the files in the Govdocs corpus as our set of target files for our experiments. A full listing of file types available in the Govdocs corpus is shown in Table 4.1 [16].

4.1.2 The Real Data Corpus

The Real Data Corpus (RDC), is a collection of disk images extracted from storage devices (e.g., hard drives, cell phones, USB sticks) that were purchased in the secondary market in the U.S. and around the world [17]. These drives are used by forensic researchers to simulate the type of data that may be found when conducting a real-world investigation. We use 1,530 RDC drives for our set of searched media. Since the RDC contains drives that originate from both U.S. and non-U.S. citizens, we opted to only use the non-U.S. drives in our experiments to mitigate data privacy concerns.

Count	Extension	Count	Extension	Count	Extension	Count	Extension
231,512	pdf	10,098	log	254	txt	14	wp
190,446	html	7,976	unk	213	pptx	8	sys
108,943	jpg	5,422	eps	191	tmp	7	dll
83,080	text	4,103	png	163	docx	5	exported
79,278	doc	3,539	swf	101	ttf	5	exe
64,974	xls	1,565	pps	92	js	3	tif
49,148	ppt	991	kml	75	bmp	2	chp
41,237	xml	943	kmz	71	pub	1	squeak
34,739	gif	639	hlp	49	xbm	1	pst
21,737	ps	604	sql	44	xlsx	1	data
17,991	csv	474	dwf	34	jar		
13,627	gz	315	java	26	zip		

Table 4.1: Distribution of file extensions in the Govdocs Corpus. Source: Foster [16]

4.2 Tools and Computing Infrastructure

Our objective was to create a block hash database that we could use on searched media to learn more about filtering non-probative blocks. This section will describe the tools we used to build our block hash database, scan our searched media for matches, and verify the presence of target files on searched drives.

4.2.1 Building the Govdocs Block Hash Database

We used the following tools and computing infrastructure to build a block hash database for files in the Govdocs corpus:

1. **md5deep**. md5deep is used to recursively calculate MD5 hash values for files in directories and subdirectories. It supports treating a file as a series of arbitrary sized blocks and calculating the hash value for each block. [18].
2. **DFXML**. The Digital Forensics XML (DFXML) format [19] provides a standard format for sharing information across different forensics tools and organizations [11]. DFXML is supported by md5deep as an output format, and by hashdb as an input format.
3. **Hashdb**. Hashdb is a tool used to create and store databases of cryptographic hashes, with a search speed of more than a hundred thousand lookups per second [20]. It can import block hashes created by other tools (e.g., md5deep) to create a hash database.
4. **Hamming Cluster**. The Naval Postgraduate School's (NPS) Hamming Cluster is a supercomputer consisting of more than 2,000 cores and 2 TB of RAM [21].

4.2.2 Bulk_Extractor

bulk_extractor is a tool that can scan disk images, directories, or files for information relevant to forensic investigations (e.g., emails, credit card numbers, URLs). It is a particularly useful tool since it does not rely on a drive's file system or file system structures to process data, which allows it to identify information in areas of a drive that are not visible to the file system, (e.g., unallocated space). Since bulk_extractor is file system agnostic, it can be used to scan any type of digital storage media. In addition, it has a built-in module for scanning a drive using hashdb databases.

4.2.3 Expert Witness Format and Libewf

The Expert Witness Compression Format (EWF) [22] is a file format used to store digital media images, such as disk and volume images. Data from digital media can be stored across several files, referred to as segment files, and can be stored in an uncompressed or compressed format [22].

Libewf [23] is an open-source library that allows users to interact with EWF files. Included in this library is ewfexport, a utility used to convert compressed EWF files to the Raw Image Format (RAW). The RAW format is used to store a bit-for-bit copy of raw data on a disk or volume [24].

4.2.4 The Sleuth Kit (TSK)

We used the following TSK tools to extract files from RDC drives:

1. **Tsk_loaddb**. Tsk_loaddb is a utility that takes a drive image and saves the image, volume, and file metadata to an SQLite database [25].
2. **mmls**. mmls displays the layout of partitions on a drive [26].
3. **icat**. icat takes an inode number and outputs the contents of a file to standard out [27].
4. **blkls**. blkls outputs file system blocks, such as NTFS and FAT data clusters, given a start-stop range and offset into a file system partition [28].

4.2.5 Examining Raw Data

We used the following tools to examine blocks in the Govdocs corpus and raw data on RDC drives:

1. **xxd**. xxd takes an input and creates a hex dump of that input, along with the ASCII representation of the data [29]
2. **dd**. The dd utility takes an input and copies a specified number of bytes to an output file [30].

4.2.6 File Type Inspectors

We used the following tools to examine the control and data structures present in different file formats:

1. **OffVis**. OffVis [31] is a Microsoft Office visualization tool for .doc, .ppt., and .xls Microsoft files. It can parse Microsoft binary format files and display data structures as well as values, offset locations, sizes, and data types for each structure.
2. **ASTiffTagViewer**. ASTiffTagViewer [32] is a tiff file format inspector that can parse tags in tiff files (e.g., tag code, data type, count, and value).

4.3 Experiment Overview

Our experiment can be summarized as follows:

1. Create a hashdb database (govdocs.hdb) of 4096-byte block hashes from files in the Govdocs corpus.
2. Run bulk_extractor against the 1,530 RDC drives, looking for sector matches to govdocs.hdb.
3. Examine the matches and determine whether or not they are probative for the target file they matched to.
4. Eliminate non-probative matches from consideration.

To perform steps 3 and 4, we conducted seven experiments in two phases. For the first phase, we examined matches for all file types. For the second phase we narrowed our test set to just JPEG files. In each experiment, we examine the matches returned by bulk_extractor and determine how many matches are true positive matches to files in the Govdocs corpus. We also conduct an analysis of the blocks causing false positive matches to Govdocs files. Our experiments follow an iterative process in which our analysis of false positives is incorporated into the design of each subsequent experiment. The 7 experiments are outlined below:

Phase 1: General Hash-based Carving (all File Types):

- *Experiment 1: Naïve Sector Matching*. For this experiment we consider all matches initially returned by bulk_extractor when conducting an analysis of true and false positive matches to Govdocs files.
- *Experiment 2: Sector Matching with rule-based non-probative block filter*. We filter non-probative blocks by using the *ad hoc* rules (see Section 4.6.3) before conducting our analysis of true and false matches.
- *Experiment 3: Sector Matching with entropy-based non-probative block filter*. We use an entropy threshold (see Section 4.6.4) to filter non-probative blocks before conducting an analysis of true and false matches to Govdocs files.
- *Experiment 4: Sector Matching with a modified rule-based non-probative block filter*. In our last Phase I experiment, we modify the histogram rule (see Section 4.6.3), and use it to filter non-probative blocks before conducting an analysis of true and false positive matches to files in the Govdocs corpus.

Phase 2: Hash-Based Carving of JPEGs:

- *Experiment 5: Sector Matching with entropy-based non-probative block filter on RDC image AT001-0039*. We examine entropy values ranging from 0–12 in order to find an entropy threshold that has the highest precision, true positive rate, and accuracy when it comes to identifying matches to Govdocs JPEG files on RDC drives.
- *Experiment 6: Sector Matching with entropy-based non-probative block filter on RDC image TR1001-0001*. We repeat the experiment described above with a second drive to help us establish our entropy threshold.
- *Experiment 7: Sector Matching with entropy-based non-probative block filter on 1,530 Drives in the RDC*. After selecting an entropy threshold of 10.9, we filter non-probative matches that do not meet this threshold before conducting an analysis of true and false matches to Govdocs JPEG files.

4.4 Building the Block Hash Database with Hashdb

To build a block hash database out of the Govdocs corpus, we used Hashdb [20]. The first step in creating the Govdocs hashdb database was to hash every file found in Govdocs. In order to do this, we used the md5deep command [18] to recursively go through all 1000 directories in the Govdocs corpus, calculate block hashes for every file in each directory, and report the output in the Digital Forensics XML (DFXML) format [19]. For each file in each directory, we calculated MD5 hashes in piecewise chunks every 4096 bytes. Garfinkel has found that most drives align sectors at 512-byte or 4096-byte boundaries [2]. Most modern filesystems, however, have shifted towards storing data in 4096 byte sectors [4]. Since a 4096 byte sector can be thought of as eight adjacent 512-byte sectors, we opted to use 4096-byte blocks [2]. This also has the added benefit of decreasing the size of our database by 1/8 without losing much of the resolution provided by 512-byte blocks, since most target files will be larger than 4096 bytes [4]. However, it is also possible that many target files will not be sector aligned on a disk. For example, the NTFS file system stores small files in the Master File Table [2]. We discuss how bulk_extractor addresses the alignment problem in Section 4.5.

In order to process multiple directories in parallel, we made use of the Naval Postgraduate School's (NPS) Hamming Cluster [21]. One DFXML file was generated for each directory and was imported into hashdb to create a block hash database. Once a block hash database had been created, we used a built-in hashdb function, *deduplicate*, to deduplicate the database so that only one instance of each hash would be found in each database. After all 1,000 hashdb databases had been created and deduplicated, we merged them all into a single hashdb database we called *govdocs.hdb*.

4.5 Scanning for Matches with BulkExtractor

Once we had a block hash database for the entire Govdocs corpus, we took our 1,530 RDC drives and scanned them for matches against our Govdocs database using bulk_extractor [33]. Since bulk_extractor is file system agnostic, it has no way of knowing where a file system starts on a drive. If we want to compare disk sectors to file blocks, however, we need to make sure that the sectors are aligned to the file blocks; otherwise, no matches to our target files will be found. Bulk_extractor deals with the alignment problem by reading eight 512-byte sectors in a 4096-byte sliding window, shifting 512-bytes and repeating for all possible alignments [4].

The 1,530 RDC drive images are stored in the Encase image format [22]. The Encase format allows digital media to be segmented into multiple, compressed evidence files. bulk_extractor is able to read the Encase headers and reconstruct the disk image. To scan each of these 1,530 drives, we ran bulk_extractor on the Hamming Cluster using the govdocs.hdb database as our set of target block hashes. For each RDC image, bulk_extractor created a report directory listing the results of scanning for the target hashes on the image. Each report directory with positive hits contains an *identifiedblocks.txt* file, listing the hash and offset in the RDC image where a match was found.

4.6 Interpreting Block Hash Matches

Since our two datasets were independently created and there is no known reason to expect to find Govdocs files on RDC drives, we initially assumed that all RDC matches for files in the Govdocs corpus were false positives. However, there were instances where we were able to verify that Govdocs files were actually present on RDC drives.

Scanning the Real Data Corpus drives for matches in the govdocs.hdb database can result in three possible scenarios: [4]:

1. None of the sectors in an RDC drive match a block in the govdocs database.
2. An RDC drive contains matches for all of the block hashes of a particular target file.
3. An RDC drive contains matches for some of the block hashes of a particular target file.

The first scenario indicates that the target file is likely not present on the drive being analyzed. However, it is possible that the file resides on the drive but has been encrypted, compressed, or obfuscated with some other encoding scheme.

The second scenario indicates that the file is present entirely on the target drive and can be reconstructed. While we did observe matches as high as 99%, we did not observe any cases where an RDC drive matched all of the block hashes of a particular file. This most likely occurred because the length of the Govdocs files to which the sector hashes matched was not an exact multiple of 4096. Out of the nearly 1 million files in the Govdocs corpus, there were only 88 files with a size that was exactly divisible by 4096. For the rest of the files, a 100% match was not possible. When constructing govdocs.hdb using md5deep, we only included blocks that were exactly 4096 bytes (i.e., we discarded the ends of the files.

The third scenario is the most challenging to interpret. One explanation for only finding some of the blocks of a Govdocs file on an RDC drive is that the file was once present but has been deleted and overwritten by new data. Another explanation is that the drive has been corrupted, and only part of the file remains. This is of particular concern with our sample of Real Data Corpus drives. Since these drives were purchased in the secondary market, the drives are used and may be damaged [17]. A third explanation is that the sector matches are the result of blocks found in the target file that are shared across multiple files. These common blocks occur when file formats, such as JPEGs, contain data structures that are found in many files of the same type. In cases where only some blocks are found, we need to determine whether the matching blocks are probative or not.

A probative block is a block that indicates with a high likelihood that a target file is or was once present on a drive [4]. These blocks do not occur in any other file, except as a copy of the original. Young et al [12] had previously called these blocks distinct, but Garfinkel and McCarrin found that blocks that appeared to be distinct in one dataset were shown to be not distinct once larger data sets were considered [4].

The majority of our matches fell under the third scenario. When only some matches to a Govdocs file are returned, we cannot be certain if the Govdocs file is really present on an RDC drive or not. One surprising observation was that we identified a Govdocs file that was over 75% present on an RDC drive (734918.pdf), but that turned out to be a false positive. Due to the uncertainty associated with the third scenario, we required further analysis of the matches returned to blocks in Govdocs files and the drives we found them on.

4.6.1 Extracting Files from RDC Drives

To help us determine whether matching sectors on RDC drives were probative or not, we first needed to know which Govdocs file the hash matched to. Hashdb can read the output file reported by bulk_extractor and associate the hash matched to a file in the target database. We also needed to know if the Govdocs file was actually present on the RDC drive. However, since bulk_extractor is file system agnostic, it only reports the byte offset in the RDC image where the match was found, not the filename.

To extract files from RDC drives, we made use of various tools. Some tools did not work with the compressed EWF format that RDC images were saved in, so we first needed to convert EWF files to to the RAW format using ewfexport [23]. Once we converted RDC drive images to RAW, we were able to use tsk_loaddb to create an SQLite database for each RDC drive we examined. From there, we used a Python script to query the database and pull out the filenames associated with matched sector hashes as well as other useful metadata such as inode numbers. Inodes were particularly useful, since we could use the inode number to extract a file from the drive using icat [27]. By using this contextual information about our matches, we were able to determine if sector hashes matched to common blocks found across many files, or if they matched to probative blocks that could be used to indicate the presence of a target file.

Some drives, however, were not compatible with ewfexport or tsk_loaddb because the image was damaged or corrupted in some way. Without the metadata structures, we could not retrieve filenames or inode numbers to extract files from these RDC drives. To handle these cases, we resorted to using other TSK tools that do not rely on file system metadata, but instead look at the data units on a drive, such as NTFS and FAT data clusters [34]. Using blkls [28] we were able to extract blocks of data by specifying a start and end block given an offset into a file system partition. Partition locations were determined using mmls [26]. Since we also knew which file we were trying to match to, we were able to use the dd [30] utility to clean up the data blocks returned by blkls and fully extract files from the RDC drives.

4.6.2 Evaluating Matches

For cases where we could fully extract a file from an RDC drive that was generating matches to a Govdocs file using an inode, we would confirm a true match by visually comparing the RDC file with the Govdocs file to see if they were the same. There were many cases, however, where an inode on the RDC drive was not available. For these situations, we would use xxd [29] to examine the sectors that matched to Govdocs blocks, as well as adjacent sectors that did not. Tracing through the hexdump of blocks in the Govdocs corpus and RDC sector matches allowed us to see whether the two files were the same as well as the point at which the two files stopped having the same content. We would use all information available in the hexdump to make this call, such as file headers, footers, and ASCII strings. As mentioned before, some files had high similarity (one Govdocs file matched over 75% to data in an RDC file) but were not the same. Other files were the same and we were able to extract them using dd [30] once we knew the start and end location for the RDC file based on our hexdump trace of the Govdocs file.

In addition, in order to understand portions of RDC files that were causing false positive matches to Govdocs files, we used specific file format inspectors. For example, we used OffVis, a Microsoft Office visualization tool to parse the data structures present in .doc, .ppt. and .xls files. By inspecting the portions of these files that were causing false positives, we were able to gain further insight into common blocks present across multiple files.

In the next sections, we discuss two methods we used for helping us identify the probative blocks in our set of sector hash matches. The first method involves flagging non-probative blocks using a set of 3 tests developed by Garfinkel and McCarrin, known as the *ad hoc rules* [4]. The second method involves flagging non-probative blocks by using an entropy threshold. For our tests, we exclude files that are smaller than 3 blocks. Garfinkel [2] observes that sector hashing does not work well against files made up of small blocks, since many small files in NTFS filesystems are not stored sector aligned.

4.6.3 *Ad hoc* Rules

To help us identify probative sector hash matches, we used the *ad hoc* rules developed by Garfinkel and McCarrin [4]. These rules attempt to flag common blocks shared across multiple files that do not help us determine whether a target file is or was present on a drive.

The 3 *ad hoc* rules are referred to as the histogram, ramp, and whitespace rules. Figure 4.1 shows an example of a block flagged by the histogram rule that consists of a single 2-byte value, 0x3f00 repeated throughout the block. Figure 4.2 shows an example of a block containing 32-bit ascending numbers, which was flagged by the ramp rule. Finally, Figure 4.3 consists of a block of all whitespace characters that was flagged by the whitespace rule.

Chapter 5 discusses how well the *ad hoc* rules helped us identify the probative blocks in

```
0000000:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
0000010:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
0000020:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
0000030:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
0000040:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
0000050:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
0000060:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
0000070:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
0000080:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
0000090:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
00000a0:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
00000b0:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
00000c0:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
00000d0:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
00000e0:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
00000f0:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
0000100:  3f00 3f00 3f00 3f00 3f00 3f00 3f00 3f00  ?.?.?.?.?.?.?.?.
```

Figure 4.1: This block matched to a Tagged Image File Format (tiff) image contained in an Encapsulated Postscript (eps) file in the Govdocs corpus. The block contains a single 2-byte value, 0x3f00 repeated throughout the block.

```
0000000:  8100 0000 8200 0000 8300 0000 8400 0000  ................
0000010:  8500 0000 8600 0000 8700 0000 8800 0000  ................
0000020:  8900 0000 8a00 0000 8b00 0000 8c00 0000  ................
0000030:  8d00 0000 8e00 0000 8f00 0000 9000 0000  ................
0000040:  9100 0000 9200 0000 9300 0000 9400 0000  ................
0000050:  9500 0000 9600 0000 9700 0000 9800 0000  ................
0000060:  9900 0000 9a00 0000 9b00 0000 9c00 0000  ................
0000070:  9d00 0000 9e00 0000 9f00 0000 a000 0000  ................
0000080:  a100 0000 a200 0000 a300 0000 a400 0000  ................
0000090:  a500 0000 a600 0000 a700 0000 a800 0000  ................
00000a0:  a900 0000 aa00 0000 ab00 0000 ac00 0000  ................
00000b0:  ad00 0000 ae00 0000 af00 0000 b000 0000  ................
00000c0:  b100 0000 b200 0000 b300 0000 b400 0000  ................
00000d0:  b500 0000 b600 0000 b700 0000 b800 0000  ................
00000e0:  b900 0000 ba00 0000 bb00 0000 bc00 0000  ................
00000f0:  bd00 0000 be00 0000 bf00 0000 c000 0000  ................
0000100:  c100 0000 c200 0000 c300 0000 c400 0000  ................
```

Figure 4.2: This block matched to a Microsoft Word document in the Govdocs corpus. The block contains an ascending sequence of 16-bit numbers padded to 32-bits by zeros.

our set of matches.

4.6.4 Entropy Threshold for General Hash-based Carving

The second method we used for identifying probative sector hash matches was to flag all blocks that fell below a fixed entropy threshold. To develop our threshold, we began by examining the entropy distribution of blocks in the Govdocs Corpus (Figure 4.4). For our experiments we used 2-byte shannon entropy values [35]. Blocks in the Govdocs corpus have an average entropy value of about 8.9, and a median entropy value of 10.76.

```
0000000:  2020 2020 2020 2020 2020 2020 2020 0d0a                ..
0000010:  2020 2020 2020 2020 2020 2020 2020 2020
0000020:  2020 2020 2020 2020 2020 2020 2020 2020
0000030:  2020 2020 2020 2020 2020 2020 2020 2020
0000040:  2020 2020 2020 2020 2020 2020 2020 2020
0000050:  2020 2020 2020 2020 0d0a 2020 2020 2020          ..
0000060:  2020 2020 2020 2020 2020 2020 2020 2020
0000070:  2020 2020 2020 2020 2020 2020 2020 2020
0000080:  2020 2020 2020 2020 2020 2020 2020 2020
0000090:  2020 2020 2020 2020 2020 2020 2020 2020
00000a0:  2020 0d0a 2020 2020 2020 2020 2020 2020    ..
00000b0:  2020 2020 2020 2020 2020 2020 2020 2020
00000c0:  2020 2020 2020 2020 2020 2020 2020 2020
00000d0:  2020 2020 2020 2020 2020 2020 2020 2020
00000e0:  2020 2020 2020 2020 2020 2020 0d0a 2020              ..
00000f0:  2020 2020 2020 2020 2020 2020 2020 2020
0000100:  2020 2020 2020 2020 2020 2020 2020 2020
```

Figure 4.3: This block matched to Extensible Metadata Platform (XMP) structure inside a Govdocs JPEG file. The block contains whitespace characters 0x20 (space), 0x0d (carriage return), and 0x0a (newline).

We considered entropy thresholds ranging from 6–10 and examined how many files would have over 90% of their blocks flagged by each entropy threshold being considered. As Garfinkel and McCarrin [4] caution, filtering too many probative blocks results in being unable to successfully identify target files. Table 4.2 shows the effects of each entropy threshold.

Percent	Ent<6	Ent<7	Ent<8	Ent<9	Ent<10
90–100	39,567	108,278	260,358	419,785	427,864
80–90	25,415	38,111	43,606	10,607	10,818
70–80	29,904	36,255	41,371	10,034	13,354
60–70	26,535	29,162	33,208	10,627	13,734
50–60	22,769	20,735	20,040	10,221	14,616
40–50	27,142	27,777	29,150	18,704	26,120
30–40	55,307	60,595	48,446	34,156	41,559
20–30	56,080	61,601	46,147	42,761	54,157
10–20	127,254	129,539	85,812	85,734	89,801
00–10	455,552	353,472	257,387	222,896	173,502

Table 4.2: The first column shows bins that correspond to the percent of a file's blocks that have been flagged by each entropy threshold. The other columns show the number of files that fall into each bin for each threshold. We examine a total of 865,525 files. Files made up of less than three 4096-byte blocks are excluded from this table.

Since we do not want to eliminate 12.5% of files from consideration, we limited our analysis to an entropy threshold of 6 for our sector hash matching experiments that involved all

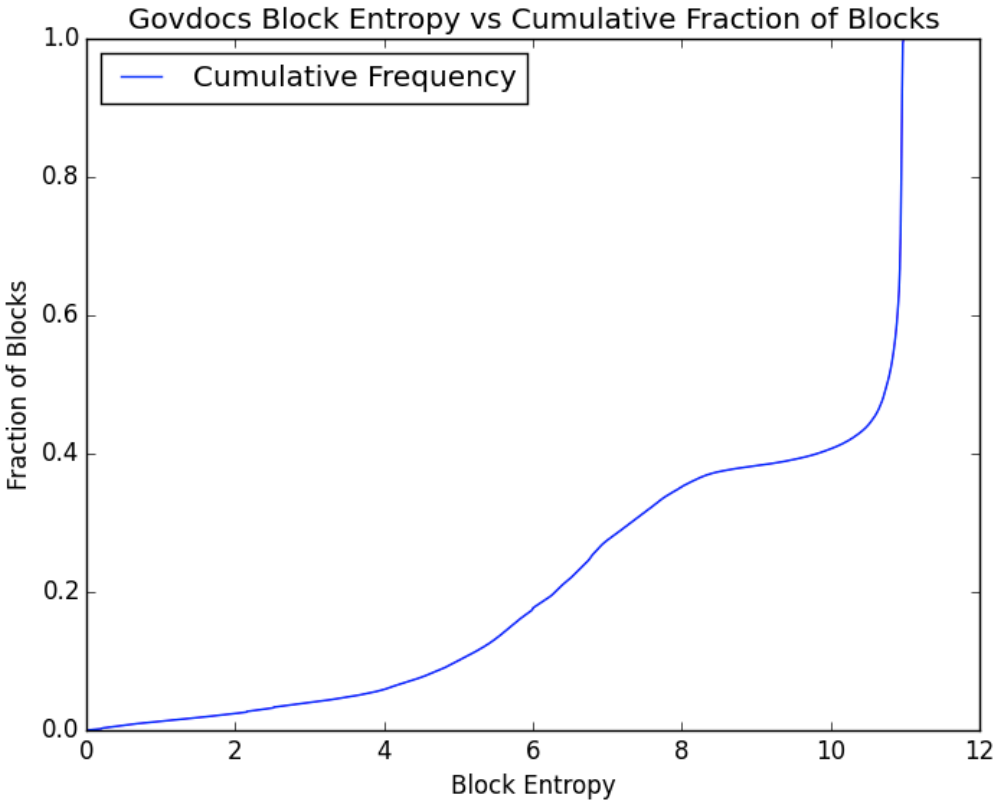


Figure 4.4: Cumulative distribution function showing the entropy distribution of blocks in the Govdocs corpus.

file types. Chapter 5 discusses how well an entropy threshold of 6 helped us identify the probative blocks in our set of matches.

4.7 Iterative Experiment Design

The experiments outlined in Section 4.3 build upon each other. Each subsequent experiment was designed after an analysis of the results generated by the previous experiment. After several iterations of this process, we realized that the general hash-based carving problem was too difficult. This led to the creation of our second phase of experiments where we focus on a single file type, since it is easier to focus on one file format instead of trying to tackle the whole problem of file identification in general.

We chose to focus on JPEG files because the JPEG format is a popular format that is highly relevant to forensic investigations, since JPEGs are widely used by individuals and organizations. Among the Govdocs corpus, JPEG files had the third highest count, accounting for about 11 percent of all files in the corpus. See Table 4.1 for a distribution of file formats in the Govdocs corpus. In addition, JPEG files have a higher than average entropy, which we believed would work well with entropy thresholds for flagging non-probative blocks. Figure 4.5 shows the block entropy distribution for JPEG file blocks. In Section 5.11, we discuss the results of using entropy thresholds to identify Govdocs JPEG files on 1,530 RDC drives.

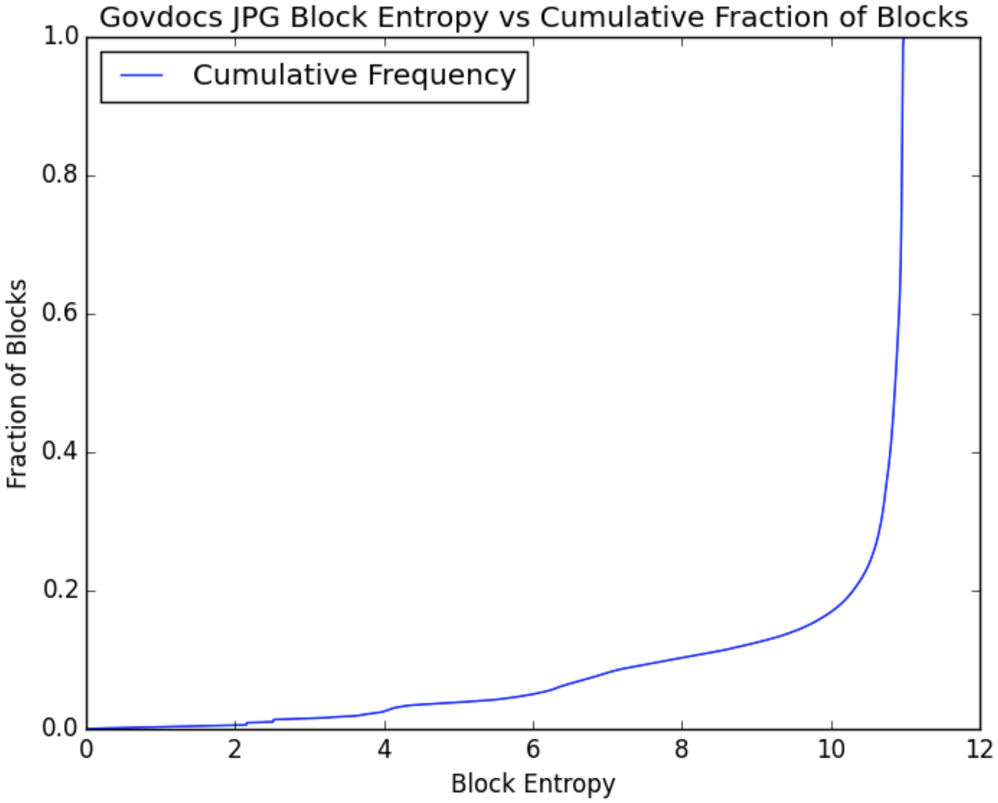


Figure 4.5: Cumulative distribution function showing the entropy distribution JPEG file blocks in the Govdocs corpus. We can see that more than 80% of JPEG blocks had an entropy value greater than 10.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Results

In this chapter, we discuss the results of running bulk_extractor against our set of 1,530 Real Data Corpus (RDC) drives for matches to *govdocs.hdb*, our database of sector hashes for target files in the Govdocs corpus. First, we discuss the matches obtained without filtering for non-probative blocks. Next, we discuss the results of filtering matches with the *ad hoc* rules and entropy thresholds. We then compare the two methods and evaluate their performance based on the number of non-probative blocks filtered and the number of probative blocks suppressed. Finally, we examine how well entropy-based non-probative block filters perform when narrowing our search to JPEG files.

5.1 Phase I: General Hash-based Carving (all File Types)

We began our experiments with the assumption that all RDC matches for files in the Govdocs corpus were false positives. However, as our experiments progressed, we learned that there were Govdocs files that were actually present on RDC drives. We empirically determined that these true matches were those where 90% or more of a Govdocs file was present on an RDC drive (see Table 5.1). For our general hash-based carving experiments, we define ground truth as follows: a target file is considered present if the drive contains over 90% of the target file's blocks in sequence; otherwise we assume that the target file is not present. Using this rule, we identified 116 files in the Govdocs corpus that are actually present on our set of 1,530 RDC drives. In addition, we tentatively assume that all blocks that match to our set of true positive files are probative. We do not expect all of these matches to be truly probative, but this method provides us with a lower bound on the number of common blocks not flagged by our classifiers.

5.2 Experiment 1: Naïve Sector Matching

For this experiment, we classify a positive as a target file that has any block match to a sector on a drive. We classify a negative as a target file where no sectors match a block in the target file.

Govdocs File	RDC Drive	RDC Matched Blocks	Govdocs File Blocks	% Matched
864164.jpg	TR1001-0001	336	337	99.7%
877715.jpg	TR1001-0001	275	276	99.6%
733719.jpg	TR1001-0001	124	125	99.2%
369594.jpg	TR1001-0001	63	64	98.4%
580812.swf	TR1001-0001	61	62	98.4%
882039.jpg	TR1001-0001	40	41	97.6%
127839.jpg	TR1001-0001	27	28	96.4%
382935.jpg	TR1001-0001	25	26	96.2%
752484.jpg	TR1001-0001	22	23	95.7%
583453.swf	TR1001-0001	15	16	93.8%
775682.jpg	TR1001-0001	12	13	92.3%

Table 5.1: Table shows 11 Govdocs files that were over 90% present and fully recoverable on the Real Data Corpus drive, TR1001-0001.

Bulk_extractor reported 7,819,881 total matches after scanning the 1,530 RDC drives for hits to sector hashes in our Govdocs database. Of these matches, 202,344 were unique sector hashes that matched to 18,847 files in the Govdocs corpus. Table 5.2 shows the top 50 sector hash matches by count; these account for about 50% of the total matches. Table 5.3 shows a confusion matrix for the file matches returned when no block filtering was in place.

Our initial prediction about the top 50 matches was that they were not probative due to their high frequency of occurrence. We believed that these matches would most likely be common data blocks found across many files, such as a block of all NULLs (0x00). We manually examined the blocks corresponding to each sector hash match and confirmed that all 50 were not probative, that is, none of the matches could be used to demonstrate that the target files the matches came from were on a drive. We found that the matches came from blocks that had been programmatically generated and that were common across multiple files. Most of these blocks could be classified as blocks of single repeating values, repeating n-grams, or pieces of font structures. Table 5.4 shows a description of the top 50 blocks.

The most seen sector hash match, *cc985e1d7a59e9917a303925bb20a2b7*, matched an Encapsulated PostScript (eps) file in the Govdocs corpus, */raid/govdocs/687/687479.eps*, 307,200 bytes into the file. We examined the hexdump of 687479.eps and learned that the match consisted of a 4096-byte block of a single repeated value, 0x3f00, that had actually occurred within a Tagged Image File Format (tiff) file contained in the eps file. We

Rank	Sector Hash	Govdocs Source	Count	%of Total Matches
1	cc985e1d7a59e9917a303925bb20a2b7	/raid/govdocs/687/687479.eps	2,112,860	27.02%
2	0336fe5fb011d886e3fc078120a53377	/raid/govdocs/244/244578.ppt	731,014	9.35%
3	fea5c666b8d94bd1e72cb4cd03ae2d66	/raid/govdocs/023/023278.doc	269,378	3.44%
4	3a504d1dcd1623cb779ed3fcfffbfdab	/raid/govdocs/656/656479.ppt	111,567	1.43%
5	97149adf997695140e3fcb31c1100776	/raid/govdocs/024/024043.ppt	111,559	1.43%
6	72e8c783a6656cb50528e6281bd680bb	/raid/govdocs/024/024043.ppt	110,911	1.42%
7	a948038b42db285a99fd310f2dc3ccd1	/raid/govdocs/923/923551.jpg	104,082	1.33%
8	a87c19f66c8899535ed0a375c54b13e0	/raid/govdocs/468/468215.xls	60,961	0.78%
9	59576cae5bf4a3233065a52d7bf5d7ee	/raid/govdocs/180/180001.doc	40,283	0.52%
10	1141af71f2f9a1c1711609b4e6e9f2be	/raid/govdocs/185/185903.doc	36,118	0.46%
11	197a76f0b5d2cde86e5696699196fd0d	/raid/govdocs/678/678724.eps	27,135	0.35%
12	2e1f1db4554d0f31ca5d4244112c59fb	/raid/govdocs/994/994081.doc	19,455	0.25%
13	7f641a3991cfee311cb49012ac28d284	/raid/govdocs/055/055142.doc	18,121	0.23%
14	7fcad0052ea504b1ad3a08e7c6d63105	/raid/govdocs/055/055142.doc	14,566	0.19%
15	5647a235945a7286436d75ba8616859b	/raid/govdocs/674/674507.eps	10,899	0.14%
16	a4277d14b330fa3b10b3507986349844	/raid/govdocs/229/229075.xls	10,847	0.14%
17	14be16700aa3bad62809154e8df0842e	/raid/govdocs/055/055142.doc	10,461	0.13%
18	4cb46c63e664f4bd9581a5a465e05580	/raid/govdocs/166/166420.text	8,258	0.11%
19	17fb70af9a4ad363c9b807779c0cb4dc	/raid/govdocs/535/535579.swf	7,588	0.10%
20	94afd4fd6f4f1db99d063e7a57839de2	/raid/govdocs/037/037634.ppt	7,407	0.09%
21	275d31cb45a7ac3be4b54d1ca642a9eb	/raid/govdocs/037/037634.ppt	7,403	0.09%
22	61ea1b5d782e844571979ca7b85e175c	/raid/govdocs/592/592551.ppt	6,899	0.09%
23	7f99c4c8f14027aafcb2a1487d7bf68e	/raid/govdocs/595/595107.pdf	6,119	0.08%
24	f07c2f8cc366fb35b494aeac5f4dbc98	/raid/govdocs/569/569152.pdf	5,555	0.07%
25	a5b55cabb7d9edc1334c294803fd0ff4	/raid/govdocs/569/569152.pdf	5,543	0.07%
26	67fd1ba632407fce9a5f017b2bf4a49c	/raid/govdocs/569/569152.pdf	5,537	0.07%
27	bdf58f11c668b2bf84bf6fcb06142d8c	/raid/govdocs/569/569152.pdf	5,536	0.07%
28	1f4176d49e00d134cc3eb6e0e3524a0d	/raid/govdocs/569/569152.pdf	5,534	0.07%
29	c21b0132c044def07944a8544285969d	/raid/govdocs/569/569152.pdf	5,533	0.07%
30	efb065c43ec215c8e57a043d64bec19d	/raid/govdocs/569/569152.pdf	5,531	0.07%
31	a9b5f5cb6f618509d38430d07fc79fdb	/raid/govdocs/569/569152.pdf	5,527	0.07%
32	c09b860194e55bb50abe83f51074dc2a	/raid/govdocs/569/569152.pdf	5,527	0.07%
33	bc76bcacc8a56e0c020b0cb4932826a2	/raid/govdocs/569/569152.pdf	5,526	0.07%
34	912c612ce5efe5d8495e38182838fb04	/raid/govdocs/569/569152.pdf	5,522	0.07%
35	97327bbcae50b4eb1c10131f30477147	/raid/govdocs/384/384797.pdf	5,308	0.07%
36	618fb335f96220dc53ff5007c8c181ec	/raid/govdocs/970/970013.pdf	5,300	0.07%
37	a8ed8e0f1f503f6309527823706fd6b2	/raid/govdocs/970/970013.pdf	5,295	0.07%
38	e27fa81e36e933db0587a420d7d80433	/raid/govdocs/970/970013.pdf	5,289	0.07%
39	25fd9d55b6d5fc9684334d86dd5445b5	/raid/govdocs/661/661889.ppt	5,264	0.07%
40	96dcc9aea1c5e9924a449abd8b8a4ee6	/raid/govdocs/656/656483.ppt	4,931	0.06%
41	d500f394824a413c9423df33610c6782	/raid/govdocs/050/050386.rtf	4,307	0.06%
42	24253c6f3198f9c7fcd815fbc15d97f1	/raid/govdocs/050/050386.rtf	4,258	0.05%
43	22a7e72e82d2f0c87d9e67c75be88d8d	/raid/govdocs/050/050386.rtf	4,245	0.05%
44	4b0745c03f76cdb0402f813881f98cda	/raid/govdocs/050/050386.rtf	4,243	0.05%
45	e163ba7a62eba9faf4b1927e733677c9	/raid/govdocs/498/498339.rtf	4,243	0.05%
46	16a276144902e299a3381cdd40c26a9d	/raid/govdocs/050/050386.rtf	4,219	0.05%
47	fdced9c106de3a16d91d225fd452766b	/raid/govdocs/050/050386.rtf	4,219	0.05%
48	6f52ebcc9f4e77efd074157ebf6dcc1f	/raid/govdocs/050/050386.rtf	4,215	0.05%
49	ad7c4c81d6279e7f91ff01149f72ec76	/raid/govdocs/050/050386.rtf	4,208	0.05%
50	2599a1aaf2a65030e1fdff5dc62df039	/raid/govdocs/050/050386.rtf	4,201	0.05%

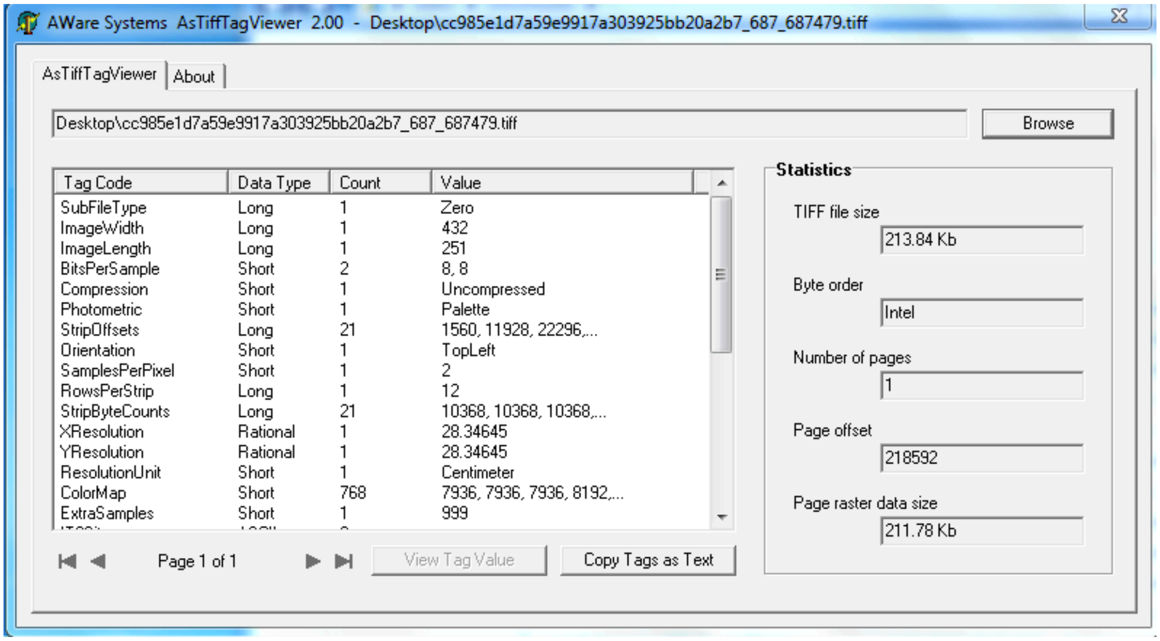
Table 5.2: Top 50 RDC Sector Hash Matches to the Govdocs Corpus. The 50 matches shown above account for 50% of total matches found.

inspected the tiff image using ASTiffTagViewer [32], as shown in Figure 5.1. According to ASTiffTagViewer, the tiff image we extracted was using a *Palette* color scheme where colors are stored in an RGB Color Map and indexed by pixel value [36]. Using a hex editor, we replaced the block of 0x3f00 values with 0xffff and confirmed that the 0x3f00 values were being used to control the background color in the tiff image. Figure 5.2 shows that we were able to modify the image by replacing parts of the background with black horizontal lines.

Table 5.4 shows that the rest of the top 50 hash matches were mostly made up of other single-value blocks, repeating n-grams, and parts of fonts. For example, the second in the

	Actual Positive	Actual Negative
Predicted Positive	116	18,731
Predicted Negative	0	896,199

Table 5.3: Confusion matrix for our naïve sector matching experiment. The accuracy for naïve sector matching was 97.95%, while the true and false positive rates were, 100% and 2.05%, respectively.


AWare Systems AsTiffTagViewer 2.00 - Desktop\cc985e1d7a59e9917a303925bb20a2b7_687_687479.tiff
AsTiffTagViewer
About
Browse
Page 1 of 1
View Tag Value
Copy Tags as Text

Desktop\cc985e1d7a59e9917a303925bb20a2b7_687_687479.tiff

Tag Code	Data Type	Count	Value
SubFileType	Long	1	Zero
ImageWidth	Long	1	432
ImageLength	Long	1	251
BitsPerSample	Short	2	8, 8
Compression	Short	1	Uncompressed
Photometric	Short	1	Palette
StripOffsets	Long	21	1560, 11928, 22296,...
Orientation	Short	1	TopLeft
SamplesPerPixel	Short	1	2
RowsPerStrip	Long	1	12
StripByteCounts	Long	21	10368, 10368, 10368,...
XResolution	Rational	1	28.34645
YResolution	Rational	1	28.34645
ResolutionUnit	Short	1	Centimeter
ColorMap	Short	768	7936, 7936, 7936, 8192,...
ExtraSamples	Short	1	999

Statistics

TIFF file size: 213.84 Kb

Byte order: Intel

Number of pages: 1

Page offset: 218592

Page raster data size: 211.78 Kb

Figure 5.1: ASTiffTagViewer parses out tags in a tiff file. The photometric tag shows that this tiff image is using a palette color scheme to index colors to an RGB color map.

list of most seen sector hash matches consists of a block of repeated values of 0x0020 while Number 9 in the list consists of a block of repeated values of 0xa56e3a. An interesting observation was that the blocks corresponding to numbers 5, 6, and 7 from the top 50 list contained the same repeating n-gram, 0x00ff00, but at different start offsets. Blocks 41–50 also shared a repeating n-gram at different start offsets. We also observed blocks with no discernible pattern. For example, blocks 23–38 consisted of random-looking values. However, upon further inspection of adjacent blocks in the files they matched to, we learned that blocks 23–38 were part of font structures (e.g., Times New Roman, Courier New).

The top 50 matches account for just over half (50.88%) of the total matches. These blocks were programmatically generated and can be found in multiple distinct files. Since they

Rank	Sector Hash	Block Description	Entropy
1	cc985e1d7a59e9917a303925bb20a2b7	0x3f00	0.00
2	0336fe5fb011d886e3fc078120a53377	0x0020	0.00
3	fea5c666b8d94bd1e72cb4cd03ae2d66	0xff7f	0.00
4	3a504d1dcd1623cb779ed3fcfffbfdab	0x00ff00	1.58
5	97149adf997695140e3fcb31c1100776	0xff0000	1.58
6	72e8c783a6656cb50528e6281bd680bb	0x0000ff	1.58
7	a948038b42db285a99fd310f2dc3ccd1	0xcc33	0.00
8	a87c19f66c8899535ed0a375c54b13e0	0x0000 0000 0001 0000	0.81
9	59576cae5bf4a3233065a52d7bf5d7ee	0xa56e3a	1.58
10	1141af71f2f9a1c1711609b4e6e9f2be	0xecd8e9	1.58
11	197a76f0b5d2cde86e5696699196fd0d	0x5d00	0.00
12	2e1f1db4554d0f31ca5d4244112c59fb	0x0001 0000	1.00
13	7f641a3991cfee311cb49012ac28d284	0x3f00 0080	1.00
14	7fcad0052ea504b1ad3a08e7c6d63105	0x0000 c842	1.00
15	5647a235945a7286436d75ba8616859b	0xfe00	0.00
16	a4277d14b330fa3b10b3507986349844	0x1100 followed by all 0x0000	0.01
17	14be16700aa3bad62809154e8df0842e	0x4200 00c8	1.00
18	4cb46c63e664f4bd9581a5a465e05580	Repeated 0x00 except for 0x01 at block offset 0xffd	0.01
19	17fb70af9a4ad363c9b807779c0cb4dc	0x0040	0.00
20	94afd4fd6f4f1db99d063e7a57839de2	0xffccff	1.58
21	275d31cb45a7ac3be4b54d1ca642a9eb	0xffffcc	1.58
22	61ea1b5d782e844571979ca7b85e175c	10 bytes of 0x00 followed by 0xffff	0.02
23	7f99c4c8f14027aafcb2a1487d7bf68e	Arial Cyrillic font	7.78
24	f07c2f8cc366fb35b494aeac5f4dbc98	Courier New Bold font	9.79
25	a5b55cabb7d9edc1334c294803fd0ff4	Courier New Bold font	9.64
26	67fd1ba632407fce9a5f017b2bf4a49c	Courier New Bold font	9.53
27	bdf58f11c668b2bf84bf6fcb06142d8c	Courier New Bold font	8.46
28	1f4176d49e00d134cc3eb6e0e3524a0d	Courier New Bold font	8.18
29	c21b0132c044def07944a8544285969d	Courier New Bold font	10.08
30	efb065c43ec215c8e57a043d64bec19d	Courier New Bold font	9.39
31	a9b5f5cb6f618509d38430d07fc79fdb	Courier New Bold font	9.65
32	c09b860194e55bb50abe83f51074dc2a	Courier New Bold font	9.01
33	bc76bcacc8a56e0c020b0cb4932826a2	Courier New Bold font	9.39
34	912c612ce5efe5d8495e38182838fb04	Courier New Bold font	9.67
35	97327bbcae50b4eb1c10131f30477147	Times New Roman, Bold Italic font	8.04
36	618fb335f96220dc53ff5007c8c181ec	Courier New Bold Italic font	9.74
37	a8ed8e0f1f503f6309527823706fd6b2	Times New Roman font	9.96
38	e27fa81e36e933db0587a420d7d80433	Courier New Bold Italic font	9.99
39	25fd9d55b6d5fc9684334d86dd5445b5	Repeated 0xffff except for 0x0000 at offset 0x79e	0.01
40	96dcc9aea1c5e9924a449abd8b8a4ee6	Repeated 0xffff except for 0x0000 at offset 0x82e	0.01
41	d500f394824a413c9423df33610c6782	0x66 (89 times) 0x0d0a 0x66 (39 times)	0.23
42	24253c6f3198f9c7fcd815fbc15d97f1	0x66 (13 times) 0x0d0a 0x66 (115 times)	0.23
43	22a7e72e82d2f0c87d9e67c75be88d8d	0x66 (91 times) 0x0d0a 0x66 (37 times)	0.23
44	4b0745c03f76cdb0402f813881f98cda	0x66 (47 times) 0x0d0a 0x66 (81 times)	0.23
45	e163ba7a62eba9faf4b1927e733677c9	0x66 (57 times) 0x0d0a 0x66 (71 times)	0.23
46	16a276144902e299a3381cdd40c26a9d	0x66 (7 times) 0x0d0a 0x66 (121 times)	0.23
47	fdced9c106de3a16d91d225fd452766b	0x66 (81 times) 0x0d0a 0x66 (47 times)	0.23
48	6f52ebcc9f4e77efd074157ebf6dcc1f	0x66 (85 times) 0x0d0a 0x66 (43 times)	0.23
49	ad7c4c81d6279e7f91ff01149f72ec76	0x66 (71 times) 0x0d0a 0x66 (57 times)	0.23
50	2599a1aaf2a65030e1fdff5dc62df039	0x66 (73 times) 0x0d0a 0x66 (55 times)	0.23

Table 5.4: Block descriptions of the top 50 sector hash matches. Most blocks consisted of single-values, repeating n-grams, and parts of fonts. We do not consider any of these blocks to be probative since they are programmatically generated blocks that are common across multiple files.

are not probative matches, we want to flag these blocks so that they do not distract forensic investigators who are searching for target files. In order to do this, we used the *ad hoc* rules and entropy threshold methods to try to eliminate the top 50 matches and other common blocks from consideration.

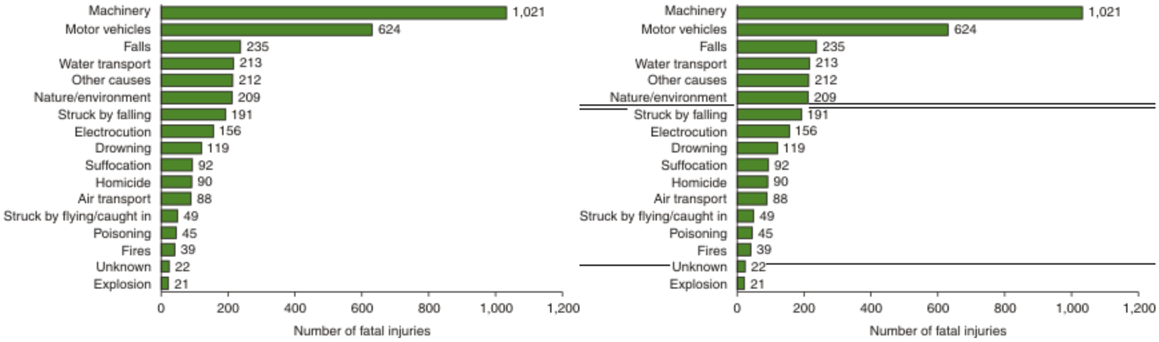


Figure 5.2: The original tiff image taken from the encapsulated postscript file appears on the left. Blocks of repeated 0x3f00 were replaced by 0xffff to produce the image on the right with the added black horizontal lines. It appears that the blocks of 0x3f00 in the tiff image were indexes into a palette color map used to produce the background color.

5.3 Experiment 2: Sector Matching with a Rule-Based Non-Probative Block Filter

For this experiment, we classify a positive as a target file that has any block match a sector on a drive, as long as the block has not been flagged as non-probative by the *ad hoc* rules. We classify a negative as a target file where no sectors match a block in the target file, or the only sectors that match are flagged by the *ad hoc* rules.

We ran the *ad hoc* rules proposed by Garfinkel and McCarrin [4] against our set of matches to see if they would flag non-probative blocks found across multiple files in our Govdocs database. Table 5.5 shows that 28,831 out of the 202,344 unique sector hash matches were flagged as non-probative, or about 14%. In addition, 6,655,930 out of the 7,819,881 total matches were flagged by the *ad hoc* rules as non-probative, eliminating about 85% of the total matches to the Govdocs corpus from consideration. However, the *ad hoc* rules also suppress some probative blocks. Our analysis of our Govdocs database revealed that the *ad hoc* rules flag over 90% of blocks for 1.65% of files in the Govdocs corpus, a measure of loss we found acceptable.

Out of the 50 top sector hash matches we described above, the *ad hoc* rules flagged 34 as non-probative. The 16 blocks that were not flagged were numbers 23–38 in Table 5.4. As Table 5.4 shows, these blocks were part of font structures such as Courier New and Times New Roman. The *ad hoc* rules failed to flag these blocks as non-probative because they

Rule	Sector Hashes Flagged	Match Count
Ramp	5,794	297,105
Space	5	251
Hist	23,288	6,361,856
Ramp+Space	0	0
Ramp+Hist	251	3,031
Space+Hist	5	251
Ramp+Space+Hist	0	0
Total	28,831	6,655,930

Table 5.5: Table shows the number of unique sector hashes that were flagged by each rule as well as the total matches flagged. Since some rules flag the same blocks as other rules, we subtract out matches that were flagged by more than one rule to avoid double counting.

```
0000000:  002f 0046 0056 021d 400d 070d 9946 d946  ./.F.V..@....F.F
0000010:  0269 1e7a 1e02 04ba 03ce 0002 ffc0 b509  .i.z............
0000020:  0d34 0202 00bd 03cb 0006 0028 03cb 0006  .4.........(....
0000030:  03cf b420 1e1e 082c b803 cb40 0f0f 231f  ... ...,...@..#.
0000040:  236f 237f 23ef 2305 2323 0813 ba03 cf00  #o#.#.#.##......
0000050:  0f03 cbb7 1840 090b 3418 181c bb03 c400  .....@..4.......
0000060:  0a00 0b03 cf40 0a60 0870 0802 0808 4438  .....@.`.p....D8
0000070:  48b8 ffc0 b509 1934 4848 43b8 ffd4 400b  H......4HHC...@.
0000080:  1f21 3444 4345 3033 3338 31bf 03bc 0030  .!4DCE03381....0
0000090:  003c 03cf 0045 03c4 0046 ffc0 b309 0e34  .<...E...F.....4
00000a0:  46b8 03c7 4009 6030 7030 0260 3001 30b8  F...@.`0p0.`0.0.
00000b0:  ffc0 b509 0c34 3030 3eb8 03cb b538 4009  .....400>....8@.
00000c0:  0c34 38b8 ffc0 b61f 2034 c038 0138 b8ff  .48..... 4.8.8..
00000d0:  c0b3 2225 3438 b8ff c0b3 3132 3438 b8ff  .."%48....1248..
00000e0:  c0b3 363a 3438 b8ff c0b3 3c3e 3438 b8ff  ..6:48....<>48..
00000f0:  c0b3 4e4f 3438 b8ff 8040 1551 5334 3851  ..NO48...@.QS48Q
0000100:  1c15 0b0b 150f 0f1a 1515 0d40 292a 340d  ...........@)*4.
```

Figure 5.3: Hexdump of part of a courier new bold font block.

do not contain easily recognizable patterns (e.g., repeating n-grams). In fact, it is difficult to imagine a rule that could be used to categorize blocks of font structures since they are usually high entropy blocks with random looking data. Figure 5.3 and Figure 5.4 show hexdumps of parts of blocks 24 and 37, which matched to Courier New Bold and Times New Roman fonts, respectively.

5.3.1 Evaluating the *Ad Hoc* Rules

The remaining 173,513 unique sector hash matches not flagged by the *ad hoc* rules matched to 9,101 files in the Govdocs corpus.

```
0000000:  3704 1bfc ac35 2719 283b 3e04 2074 1047  7....5’.(;>. t.G
0000010:  2e23 2e09 01de 1028 1920 3e1e fddb 4a3f  .#.....(. >...J?
0000020:  2d20 122a 0001 0089 ffb2 0499 074d 0007  - .*.........M..
0000030:  0082 402d 0306 0707 3b02 0314 0207 0002  ..@-....;.......
0000040:  0306 0505 4c04 0314 0404 0300 0707 4c02  ....L.........L.
0000050:  0114 0207 0602 0105 0403 0308 0709 0606  ................
0000060:  01b8 017c 4010 2803 0706 0609 0208 2801  ...|@.(.......(.
0000070:  1002 0208 0a7f b901 db00 182b 2b10 3c01  ...........++.<.
0000080:  2f2b 3c2b 10c0 0119 1239 2f00 183f 2bed  /+<+.....9/..?+.
0000090:  0039 0111 1239 1217 3987 082e 2b05 7d10  .9...9..9...+.}.
00000a0:  c487 0e2e 182b 7d10 c487 082e 182b 870e  .....+}......+..
00000b0:  7dc4 3130 0133 0101 0727 2501 044f 4afe  }.10.3...’%..OJ.
00000c0:  c8fe 10c6 2201 2d01 9507 4df8 6503 fd5b  ....".-...M.e..[
00000d0:  4097 fcc9 0001 005a ff00 0565 052f 0039  @......Z...e./.9
00000e0:  00ff 401f 5528 7526 8728 8637 8738 9628  ..@.U(u&.(.7.8.(
00000f0:  9737 9638 0869 1d01 0000 0102 0304 0507  .7.8.i..........
0000100:  0809 0aba 02ce 02cf 02d0 400d 2256 0026  ..........@."V.&
```

Figure 5.4: Hexdump of part of a times new roman font block.

After comparing the remaining 173,513 blocks that were not flagged by the *ad hoc* rules to the blocks in our set of 116 true positive files, we learned that 20,755 out of 173,513, or about 12% of the blocks, matched 116 of our true positive files. Since we are assuming that all of the block matches are probative, this means that the *ad hoc* rules failed classify the remaining 88% of the blocks as non-probative. Table 5.6 shows a confusion matrix for our experiment repeated with the *ad hoc* rules in place as a filter.

	Actual Positive	Actual Negative
Predicted Positive	116	8,985
Predicted Negative	0	905,945

Table 5.6: Confusion matrix for the *ad hoc* rules non-probative block filter experiment. The accuracy for the *ad hoc* rules was 99.02%, while the true and false positive rates were, 100% and 0.98%, respectively.

5.4 Experiment 3: Sector Matching with an Entropy-Based Non-Probative Block Filter

For this experiment, we classify a positive as a target file that has any block match a sector on a drive, as long as the block has not been flagged as non-probative by an entropy threshold of 6. We classify a negative as a target file where no sectors match a block in the target file, or the only sectors that match are below an entropy threshold of 6.

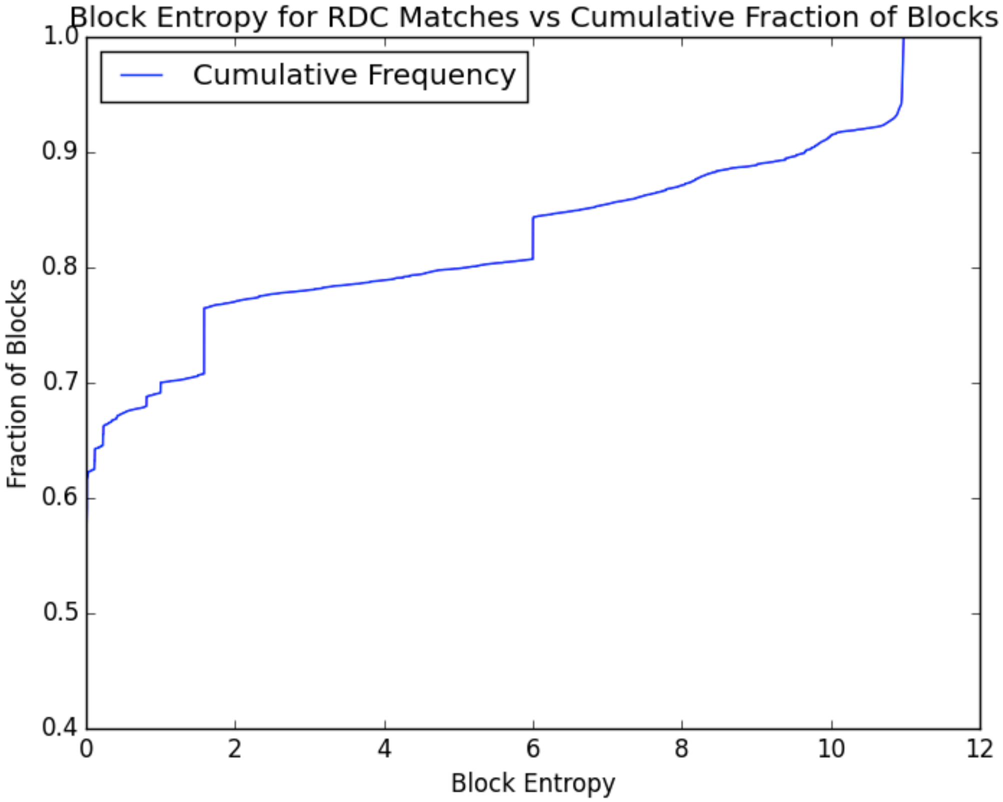


Figure 5.5: Cumulative distribution function for entropy values of RDC blocks that matched to the Govdocs corpus.

As mentioned in Section 4.6.4 blocks in the Govdocs corpus have an average entropy value of 8.9 and a median entropy value of 10.76. Figure 4.4 shows a Cumulative Distribution Function of block entropy values in the Govdocs corpus.

In contrast, for RDC blocks that matched to blocks in our Govdocs database, the average entropy value is 2.03 and the median entropy value is 0.006. Figure 5.5 shows a Cumulative Distribution Function of the entropy values for RDC blocks that match to the Govdocs database. We can see that a majority of matching blocks (69%), have an entropy value of less than 1 and that 90% of matching blocks have an entropy value of less than 10. Table 5.7 shows how many unique sector matches and total matches are flagged using different entropy thresholds.

Entropy Threshold	Blocks Flagged	% Total Blocks	Match Count	% of Total Matches
< 1	11,834	5.85%	5,406,337	69.14%
< 2	14,326	7.08%	6,024,325	77.04%
< 3	16,515	8.16%	6,103,402	78.05%
< 4	21,081	10.42%	6,170,182	78.90%
< 5	28,486	14.08%	6,249,018	79.91%
< 6	30,621	15.13%	6,312,375	80.72%
< 7	39,263	19.40%	6,685,921	85.50%
< 8	45,249	22.36%	6,813,496	87.13%
< 9	52,819	26.10%	6,950,770	88.89%
< 10	63,921	31.59%	7,152,249	91.46%
< 11	202,140	99.90%	7,817,949	99.98%
< 12	202,344	100.00%	7,819,881	100.00%

Table 5.7: Table shows the number of unique blocks that were flagged by each entropy threshold as well as the total matches flagged.

For the top 50 matches, the last column of Table 5.4 shows the block entropy values. We can see that blocks made up of 2-byte repeating values, such as Blocks 1, 2, and 3, have entropy values of zero, since there is no variation throughout the block. Blocks with repeating n-grams that are longer than 2-bytes had entropy values that ranged from 0.006 to 1.585. Blocks that were part of font structures had much higher entropy values, ranging from 7.783 to 10.076. We found that an entropy threshold of 7 flags the exact same 36 blocks from the top 50 matches as the *ad hoc* rules. Additionally, an entropy threshold of 7 flags 19% of unique block matches and 85.5% of total matches, compared to 14% of unique block matches and 85.12% of total matches flagged with the *ad hoc* rules. In order to eliminate all but one of the font blocks (number 29), we would need to use an entropy threshold of 10.

As with the *ad hoc* rules, using entropy thresholds also suppresses some probative blocks. Figure 4.2 shows that an entropy threshold of 7 would flag over 90% of blocks for 12.5% of files in the Govdocs corpus while an entropy threshold of 10 would flag over 90% of blocks for almost half of all files in the Govdocs corpus. When more than 90% of a file's blocks are flagged, the probability of carving the file is greatly reduced [4]. We find that for a general entropy rule, the number of files lost is too high for entropy thresholds of 7 and above. A more acceptable threshold level is an entropy threshold of 6, which only flags over 90% of blocks for less than 5% of files in the Govdocs corpus. An entropy threshold of 6 flags the exact same 36 blocks from our top 50 matches as the entropy threshold of 7 and the *ad hoc* rules. However, the total matches flagged with an entropy threshold of 6 falls to 80.72%, compared to 85.50% with an entropy threshold of 7, and 85.11% for the *ad hoc* rules. The percentage of unique block matches flagged was 15% for an entropy threshold of 6, compared to 19% for an entropy threshold of 7, and 14% for the *ad hoc* rules.

5.4.1 Evaluating an Entropy Threshold of 6

Since we found an entropy threshold of 7 to be too aggressive at flagging blocks, we repeated the procedure described in Section 5.3.1 with 171,723 blocks that were not flagged by an entropy threshold of 6 (i.e., blocks that had an entropy value of 6 or greater). Using the same set of 116 true positive files, we were able to match 20,495 out of the 171,723 blocks, or about 12%, to 116 true positive files. Since we are assuming that all of these matches are probative, this means that 88% of the remaining blocks are non-probative, similar to the *ad hoc* rules. Table 5.8 shows a confusion matrix for our experiment with blocks below an entropy threshold of 6 discarded. Compared to the *ad hoc* rules, an entropy threshold of 6 had a lower accuracy, lower precision, and higher false positive rate.

	Actual Positive	Actual Negative
Predicted Positive	116	10,797
Predicted Negative	0	904,133

Table 5.8: Confusion matrix for using an entropy threshold of 6 as a non-probative block filter. The accuracy for an entropy threshold of 6 was 98.82%, while the true and false positive rates were, 100% and 1.18%, respectively.

5.5 False Positives not Eliminated by the *Ad Hoc* Rules vs. Entropy Threshold

In order to learn more about the differences in the types of blocks being flagged by the *ad hoc* rules and entropy threshold methods, we examined the following types of blocks:

1. Blocks flagged by the *ad hoc* rules that had high entropy.
2. Blocks not flagged by the *ad hoc* rules that had low entropy.

We did not expect blocks with high entropy to be flagged by the *ad hoc* rules because high entropy blocks generally do not consist of obvious patterns (e.g., repeating n-grams). We also did not expect the *ad hoc* rules to fail to flag low entropy blocks since they generally do consist of obvious patterns. The highest entropy value we could find for blocks that were flagged by one of the *ad hoc* rules was an entropy value of 8.53. The lowest entropy value we could find for blocks that were not flagged by one of the *ad hoc* rules was 1.66. For the following sections, we define high entropy as a value of 8 or greater, and low entropy as a value of less than 2.

5.5.1 High Entropy Blocks Flagged by the *Ad Hoc* Rules

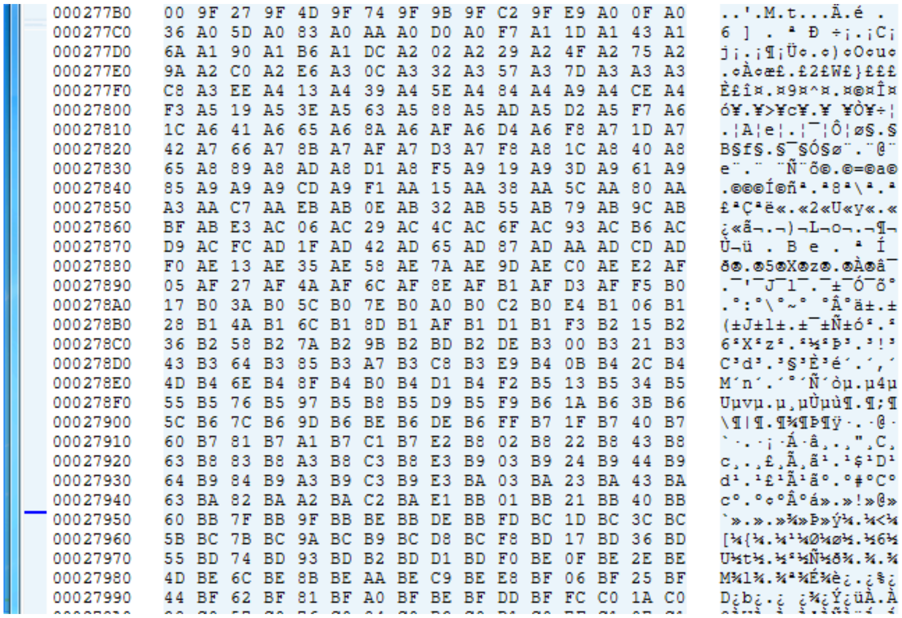
Out of the set of blocks that are flagged by the *ad hoc* rules, there are 9 blocks with an entropy value above 8. All of these blocks are flagged by the histogram rule. Table 5.9 gives a description of each block. The first block in the table contains Microsoft Office symbols, which is where the block's high entropy value comes from. Figure 5.6 shows a portion of that block. The histogram rule is triggered for this block because it contains 260 4-byte values of NULLs (probably used for padding) before reaching the start of the symbol characters. This exceeds the histogram rule's threshold of a maximum of 256 instances of a single 4-byte value (one-fourth of a 4096 byte block), so the block is flagged.

	Sector Hash	**Govdocs Source**	**Block Description**	**Entropy**
1	f613bc772c6838dc24e5b6af3dd6e21a	/raid/govdocs/718/718287.doc	Microsoft Word symbols	8.12
2	7009af9e4b945fdc8961039225ac9801	/raid/govdocs/952/952525.jpg	XMP structure, Nikon ICC Profile	8.23
3	d74d8166ea52fe4d3d525229102487be	/raid/govdocs/053/053350.jpg	XMP structure, Nikon ICC Profile	8.29
4	af0e1ae76c9b8702d0f46b0f0767c576	/raid/govdocs/588/588015.ppt	XMP structure, Nikon ICC Profile	8.03
5	2718534f9ef5c03ed379e4973d731fa5	/raid/govdocs/704/704318.ppt	Berrylishious Template	8.12
6	2590c852c36424bbd0deaaddee64e838	/raid/govdocs/717/717367.ppt	Native American Template 5	8.52
7	bada54e8fd72480054946ee99af0293c	/raid/govdocs/717/717367.ppt	Native American Template 5	8.36
8	65384f5b5784ddeea0bc86d5bb74ff29	/raid/govdocs/717/717367.ppt	Native American Template 5	8.53
9	7c3b81c3201ffb3c385fb89ab1e81e89	/raid/govdocs/717/717367.ppt	Native American Template 5	8.52

Table 5.9: Table shows the 9 blocks with entropy greater than 8 that were flagged by the histogram rule and the structures they match to.

Blocks 2–4 in Table 5.9 are all hybrid blocks containing parts of an Extensible Metadata Platform (XMP) [37] structure and parts of an International Color Consortium (ICC) profile [38]. Figure 5.7 shows a portion of block 2. The histogram rule is triggered for this block due to 269 4-byte values of 0x20 (whitespace padding) contained in the XMP structure. The high entropy comes from the rest of the block, which is an International Color Consortium (ICC) profile for NIKON devices, as can be seen in the hexdump of Figure 5.7.

Blocks 5–9 in Table 5.9 are all part of design structures for Microsoft PowerPoint Slide Templates. Figure 5.8 shows a portion of block 5 which represents the general format of these 4 blocks. These blocks are hybrid blocks containing data padded by bytes of NULLs.



```
000277B0   00 9F 27 9F 4D 9F 74 9F 9B 9F C2 9F E9 A0 0F A0   ..'.M.t...Â.é .
000277C0   36 A0 5D A0 83 A0 AA A0 D0 A0 F7 A1 1D A1 43 A1   6 ] . ª Ð ÷¡.¡C¡
000277D0   6A A1 90 A1 B6 A1 DC A2 02 A2 29 A2 4F A2 75 A2   j¡.¡¶¡Ü¢.¢)¢O¢u¢
000277E0   9A A2 C0 A2 E6 A3 0C A3 32 A3 57 A3 7D A3 A3 A3   .¢À¢æ£.£2£W£}£££
000277F0   C8 A3 EE A4 13 A4 39 A4 5E A4 84 A4 A9 A4 CE A4   È£î¤.¤9¤^¤.¤©¤Î¤
00027800   F3 A5 19 A5 3E A5 63 A5 88 A5 AD A5 D2 A5 F7 A6   ó¥.¥>¥c¥.¥ ¥Ò¥÷¦
00027810   1C A6 41 A6 65 A6 8A A6 AF A6 D4 A6 F8 A7 1D A7   .¦A¦e¦.¦¯¦Ô¦ø§.§
00027820   42 A7 66 A7 8B A7 AF A7 D3 A7 F8 A8 1C A8 40 A8   B§f§.§¯§Ó§ø¨.¨@¨
00027830   65 A8 89 A8 AD A8 D1 A8 F5 A9 19 A9 3D A9 61 A9   e¨.¨ ¨Ñ¨õ©.©=©a©
00027840   85 A9 A9 A9 CD A9 F1 AA 15 AA 38 AA 5C AA 80 AA   .©©©Í©ñª.ª8ª\ª.ª
00027850   A3 AA C7 AA EB AB 0E AB 32 AB 55 AB 79 AB 9C AB   £ªÇªë«.«2«U«y«.«
00027860   BF AB E3 AC 06 AC 29 AC 4C AC 6F AC 93 AC B6 AC   ¿«ã¬.¬)¬L¬o¬.¬¶¬
00027870   D9 AC FC AD 1F AD 42 AD 65 AD 87 AD AA AD CD AD   Ù¬ü . B e . ª Í
00027880   F0 AE 13 AE 35 AE 58 AE 7A AE 9D AE C0 AE E2 AF   ð®.®5®X®z®.®À®â¯
00027890   05 AF 27 AF 4A AF 6C AF 8E AF B1 AF D3 AF F5 B0   .¯'¯J¯l¯.¯±¯Ó¯õ°
000278A0   17 B0 3A B0 5C B0 7E B0 A0 B0 C2 B0 E4 B1 06 B1   .°:°\°~° °Â°ä±.±
000278B0   28 B1 4A B1 6C B1 8D B1 AF B1 D1 B1 F3 B2 15 B2   (±J±l±.±¯±Ñ±ó².²
000278C0   36 B2 58 B2 7A B2 9B B2 BD B2 DE B3 00 B3 21 B3   6²X²z².²½²Þ³.³!³
000278D0   43 B3 64 B3 85 B3 A7 B3 C8 B3 E9 B4 0B B4 2C B4   C³d³.³§³È³é´.´,´
000278E0   4D B4 6E B4 8F B4 B0 B4 D1 B4 F2 B5 13 B5 34 B5   M´n´.´°´Ñ´òµ.µ4µ
000278F0   55 B5 76 B5 97 B5 B8 B5 D9 B5 F9 B6 1A B6 3B B6   Uµvµ.µ¸µÙµù¶.¶;¶
00027900   5C B6 7C B6 9D B6 BE B6 DE B6 FF B7 1F B7 40 B7   \¶|¶.¶¾¶Þ¶ÿ·.·@·
00027910   60 B7 81 B7 A1 B7 C1 B7 E2 B8 02 B8 22 B8 43 B8   `·.·¡·Á·â¸.¸"¸C¸
00027920   63 B8 83 B8 A3 B8 C3 B8 E3 B9 03 B9 24 B9 44 B9   c¸.¸£¸Ã¸ã¹.¹$¹D¹
00027930   64 B9 84 B9 A3 B9 C3 B9 E3 BA 03 BA 23 BA 43 BA   d¹.¹£¹Ã¹ãº.º#ºCº
00027940   63 BA 82 BA A2 BA C2 BA E1 BB 01 BB 21 BB 40 BB   cº.º¢ºÂºá».»!»@»
00027950   60 BB 7F BB 9F BB BE BB DE BB FD BC 1D BC 3C BC   `».».»¾»Þ»ý¼.¼<¼
00027960   5B BC 7B BC 9A BC B9 BC D8 BC F8 BD 17 BD 36 BD   [¼{¼.¼¹¼Ø¼ø½.½6½
00027970   55 BD 74 BD 93 BD B2 BD D1 BD F0 BE 0F BE 2E BE   U½t½.½²½Ñ½ð¾.¾.¾
00027980   4D BE 6C BE 8B BE AA BE C9 BE E8 BF 06 BF 25 BF   M¾l¾.¾ª¾É¾è¿.¿%¿
00027990   44 BF 62 BF 81 BF A0 BF BE BF DD BF FC C0 1A C0   D¿b¿.¿ ¿¾¿Ý¿üÀ.À
```

Figure 5.6: Block containing Microsoft Office symbols

```
0000400:  2020 2020 2020 2020 2020 2020 2020 2020
0000410:  2020 2020 2020 2020 2020 2020 2020 2020
0000420:  2020 2020 2020 2020 2020 2020 2020 2020
0000430:  2020 2020 2020 2020 200a 2020 2020 2020           .
0000440:  2020 2020 2020 2020 2020 2020 2020 2020
0000450:  2020 2020 2020 2020 2020 2020 2020 2020
0000460:  2020 2020 2020 2020 2020 2020 2020 2020
0000470:  200a 3c3f 7870 6163 6b65 7420 656e 643d   .<?xpacket end=
0000480:  2777 273f 3eff e221 cc49 4343 5f50 524f  ’w’?>..!.ICC_PRO
0000490:  4649 4c45 0001 0100 0021 bc4e 4b4f 4e02  FILE.....!.NKON.
00004a0:  2000 006d 6e74 7252 4742 2058 595a 2007   ..mntrRGB XYZ .
```

Figure 5.7: This hybrid block consisting of 269 4-byte values of 0x20 exceeds the histogram rule’s threshold of 256 instances of a single 4-byte value. The 0x20 bytes are part of an Extensible Metadata Platform (XMP) structure. The rest of the block is an International Color Consortium (ICC) profile for NIKON devices.

We learned that these blocks control the background design for Microsoft PowerPoint slides after we replaced the 0x0000 values contained throughout the block with values of 0xffff. The change distorted the slide design used in the PowerPoint document. The effects of the change on the Berrylishious design template are shown in Figure 5.9. Similar effects were achieved for the Native American Template 5 in blocks 6–9.

```
00000a0:  2ae9 14ec bceb 76ae fbdb fb85 7fe4 7229  *.....v.......r)
00000b0:  6b7f 4667 4b7f bb32 e3b9 f0e9 e63f 56d2  k.FgK..2.....?V.
00000c0:  f5c9 4ae4 afe5 bc45 a3d9 31ab 93e7 d9c4  ..J....E..1.....
00000d0:  ab20 0000 0000 0000 0000 0000 0000 0000  . ..............
00000e0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
0000100:  0000 0000 0000 0000 0000 0000 0000 0000  ................
0000110:  0000 0000 0000 0000 0000 0000 0000 0000  ................
0000120:  0000 0000 0000 0000 0000 0000 0000 0000  ................
0000130:  0000 0000 0000 0000 0000 0000 0000 0000  ................
0000140:  0000 0227 ac03 97cc aeec 575c bc37 ea6f  ...’......W\.7.o
0000150:  0ed8 7bcf 1b99 11f8 33e1 f24f f4a9 33fa  ..{.....3..O..3.
```

Figure 5.8: This hybrid block consists of data separated by NULL bytes of padding. The histogram rule is triggered for the block because the block contains more than 256 4-byte values of 0x00.

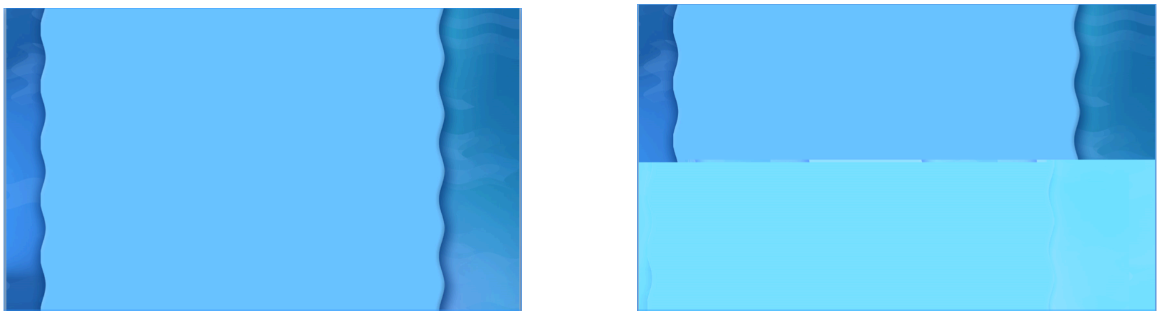


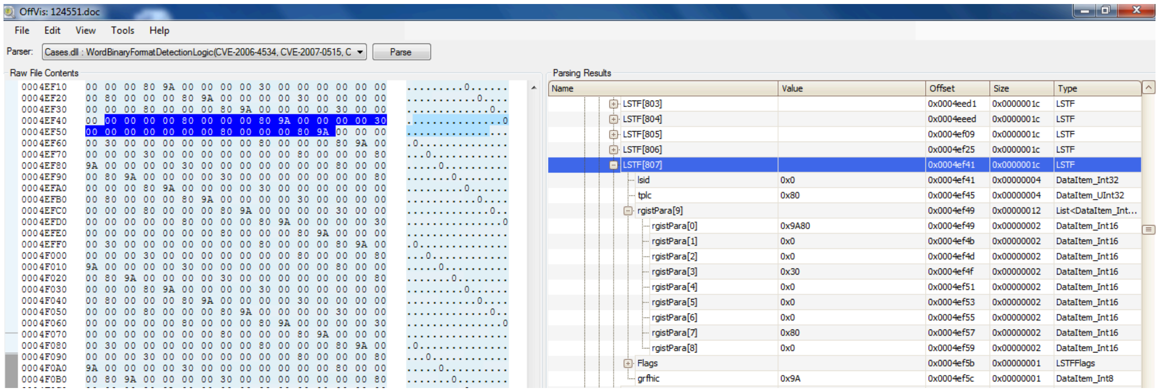
Figure 5.9: The original Berrylishious slide design is shown on the left. After replacing NULL bytes of padding with 0xff, the distorted image on the right was produced.

5.5.2 Low Entropy Blocks not Flagged by the *Ad Hoc* Rules

There are no blocks with an entropy value below 1 that are not flagged by the *ad hoc* rules. There are 284 unique RDC matches to the Govdocs database that are not flagged by any rule, but that have entropy values of less than 2. We manually examined 10 of these blocks and observed that all 10 had the same general pattern, shown in Figure 5.10. The pattern is a repeating 18-byte n-gram found in Microsoft Office Word files. We examined this pattern using OffVis [31], a Microsoft Office visualization tool, and learned it is part of a Microsoft Word list style structure, or LSTF, used in paragraph formatting [39]. Microsoft Word LSTF structures contain an 18-byte array, rgistdPara, that specifies the style that is linked to a list [40]. Figure 5.11 shows how we used OffVis to parse the rgistPara array inside our matched blocks.

```
0000000:  0000 0000 0000 0080 0000 0080 9a00 0000  ................
0000010:  0030 0000 0000 0000 0080 0000 0080 9a00  .0..............
0000020:  0000 0030 0000 0000 0000 0080 0000 0080  ...0............
0000030:  9a00 0000 0030 0000 0000 0000 0080 0000  .....0..........
0000040:  0080 9a00 0000 0030 0000 0000 0000 0080  .......0........
0000050:  0000 0080 9a00 0000 0030 0000 0000 0000  .........0......
0000060:  0080 0000 0080 9a00 0000 0030 0000 0000  ...........0....
0000070:  0000 0080 0000 0080 9a00 0000 0030 0000  .............0..
0000080:  0000 0000 0080 0000 0080 9a00 0000 0030  ...............0
0000090:  0000 0000 0000 0080 0000 0080 9a00 0000  ................
```

Figure 5.10: This block consists of a repeating 18-byte pattern. The pattern is part of a Microsoft Word LSTF paragraph formatting structure.



Parser: Cases.dll : WordBinaryFormatDetectionLogic(CVE-2006-4534, CVE-2007-0515, C

```
0004EF10   00 00 00 80 9A 00 00 00 00 30 00 00 00 00 00 00   .........0......
0004EF20   00 80 00 00 00 80 9A 00 00 00 00 30 00 00 00 00   ...........0....
0004EF30   00 00 00 80 00 00 00 80 9A 00 00 00 00 30 00 00   .............0..
0004EF40   00 00 00 00 00 80 00 00 00 80 9A 00 00 00 00 30   ...............0
0004EF50   00 00 00 00 00 00 00 80 00 00 00 80 9A 00 00 00   ................
0004EF60   00 30 00 00 00 00 00 00 00 80 00 00 00 80 9A 00   .0..............
0004EF70   00 00 00 30 00 00 00 00 00 00 00 80 00 00 00 80   ...0............
0004EF80   9A 00 00 00 00 30 00 00 00 00 00 00 00 80 00 00   .....0..........
0004EF90   00 80 9A 00 00 00 00 30 00 00 00 00 00 00 00 80   .......0........
0004EFA0   00 00 00 80 9A 00 00 00 00 30 00 00 00 00 00 00   .........0......
0004EFB0   00 80 00 00 00 80 9A 00 00 00 00 30 00 00 00 00   ...........0....
0004EFC0   00 00 00 80 00 00 00 80 9A 00 00 00 00 30 00 00   .............0..
0004EFD0   00 00 00 00 00 80 00 00 00 80 9A 00 00 00 00 30   ...............0
0004EFE0   00 00 00 00 00 00 00 80 00 00 00 80 9A 00 00 00   ................
0004EFF0   00 30 00 00 00 00 00 00 00 80 00 00 00 80 9A 00   .0..............
0004F000   00 00 00 30 00 00 00 00 00 00 00 80 00 00 00 80   ...0............
0004F010   9A 00 00 00 00 30 00 00 00 00 00 00 00 80 00 00   .....0..........
0004F020   00 80 9A 00 00 00 00 30 00 00 00 00 00 00 00 80   .......0........
0004F030   00 00 00 80 9A 00 00 00 00 30 00 00 00 00 00 00   .........0......
0004F040   00 80 00 00 00 80 9A 00 00 00 00 30 00 00 00 00   ...........0....
0004F050   00 00 00 80 00 00 00 80 9A 00 00 00 00 30 00 00   .............0..
0004F060   00 00 00 00 00 80 00 00 00 80 9A 00 00 00 00 30   ...............0
0004F070   00 00 00 00 00 00 00 80 00 00 00 80 9A 00 00 00   ................
0004F080   00 30 00 00 00 00 00 00 00 80 00 00 00 80 9A 00   .0..............
0004F090   00 00 00 30 00 00 00 00 00 00 00 80 00 00 00 80   ...0............
0004F0A0   9A 00 00 00 00 30 00 00 00 00 00 00 00 80 00 00   .....0..........
0004F0B0   00 80 9A 00 00 00 00 30 00 00 00 00 00 00 00 80   .......0........
```

Name	Value	Offset	Size	Type
⊞ LSTF[803]		0x0004eed1	0x0000001c	LSTF
⊞ LSTF[804]		0x0004eeed	0x0000001c	LSTF
⊞ LSTF[805]		0x0004ef09	0x0000001c	LSTF
⊞ LSTF[806]		0x0004ef25	0x0000001c	LSTF
⊟ LSTF[807]		0x0004ef41	0x0000001c	LSTF
lsid	0x0	0x0004ef41	0x00000004	DataItem_Int32
tplc	0x80	0x0004ef45	0x00000004	DataItem_UInt32
⊟ rgistPara[9]		0x0004ef49	0x00000012	List<DataItem_Int...
rgistPara[0]	0x9A80	0x0004ef49	0x00000002	DataItem_Int16
rgistPara[1]	0x0	0x0004ef4b	0x00000002	DataItem_Int16
rgistPara[2]	0x0	0x0004ef4d	0x00000002	DataItem_Int16
rgistPara[3]	0x30	0x0004ef4f	0x00000002	DataItem_Int16
rgistPara[4]	0x0	0x0004ef51	0x00000002	DataItem_Int16
rgistPara[5]	0x0	0x0004ef53	0x00000002	DataItem_Int16
rgistPara[6]	0x0	0x0004ef55	0x00000002	DataItem_Int16
rgistPara[7]	0x80	0x0004ef57	0x00000002	DataItem_Int16
rgistPara[8]	0x0	0x0004ef59	0x00000002	DataItem_Int16
⊞ Flags		0x0004ef5b	0x00000001	LSTFFlags
grfhic	0x9A	0x0004ef5c	0x00000001	DataItem_Int8

Figure 5.11: The view on the left shows the repeating 18-byte pattern found in blocks that were not flagged by any rule that had an entropy value of less than 2. The view on the right shows the 18-byte rgistdPara array inside the LSTF structure which controls list styles used in Microsoft Word paragraph formatting.

We created a rule to match the rgistdPara array 18-byte pattern and examined the 274 remaining blocks. We found that 273 of these remaining blocks matched our rule, giving us a total of 283 blocks out of 284 that matched the rgistdPara array pattern. The remaining block consisted of a repeating n-gram of 0x0d0a0d0a09 and is shown in Figure 5.12. The histogram rule failed to flag this block because the histogram rule splits a 4096 byte block into consecutive 4-byte values. Since the pattern repeats every 5 bytes, the same 4-byte value is not seen until 6-bytes later. However, if the 4-byte value starts at a byte offset relative to the beginning of the block (starting from offset 0) that is not divisible by 4, it is not counted since the histogram rule only looks at 4-byte values that are aligned on a 4-byte boundary. For the block in question, we only observed 205 instances of the same 4-byte value, which failed to meet the threshold of 256 required by the histogram rule.

```
0000000:  0d0a 0d0a 090d 0a0d 0a09 0d0a 0d0a 090d  ................
0000010:  0a0d 0a09 0d0a 0d0a 090d 0a0d 0a09 0d0a  ................
0000020:  0d0a 090d 0a0d 0a09 0d0a 0d0a 090d 0a0d  ................
0000030:  0a09 0d0a 0d0a 090d 0a0d 0a09 0d0a 0d0a  ................
0000040:  090d 0a0d 0a09 0d0a 0d0a 090d 0a0d 0a09  ................
0000050:  0d0a 0d0a 090d 0a0d 0a09 0d0a 0d0a 090d  ................
```

Figure 5.12: This block consists of a repeating 5-byte pattern with an entropy value of less than 2. This pattern is not flagged by the histogram rule because the histogram rule fails to flag 4-byte values if they are not aligned on 4-byte boundaries relative to the beginning of the block.

5.6 Replacing the *Ad hoc* Rules with a Single Modified Histogram Rule

The histogram rule [4] attempts to filter out blocks with repeated 4-byte values. It reads in a 4096-byte block as a sequence of 1024 4-byte integers and computes a histogram of how many times values appear. If a value appears more than 256 times (more than one fourth of the block), the rule is triggered and the block is filtered. If there are only one or two distinct 4-byte values, the rule is also triggered.

We decided to modify this rule to see if we could improve its performance. In particular, we were concerned with finding blocks that consisted of patterns that were not aligned on 4-byte boundaries, such as the rgistdPara array described in Section 5.5.2. The first change we made was to increase the block granularity by reading 2048 2-byte values instead of 1024 4-byte values. After this change was made, we needed to increase the count threshold from 256 to 512 in order to keep the rule threshold at one fourth of the block. The original rule also had an explicit test for only one or two 4-byte values. With this test in place, the rule would trigger if there were 1024 instances of a single 4-byte value. It would also trigger if there were at least 512 instances of a 4-byte value when only two distinct values were present in the block. Since our new rule looks at 2-byte values, the corresponding test would be to check for four distinct 2-byte values or less. If there are only four distinct 2-byte values, this means that there are at least 512 instances of a single 2-byte value.

However, a count of 512 represents one half of a block when 4-byte values are being read and one fourth of a block when 2-byte values are being read. For the original histogram rule that reads 1024 4-byte values, all blocks that meet the count threshold of 256, which accounts for one fourth of the block, will also meet the count threshold for 512. Thus, explicitly testing for only one or two distinct 4-byte values in the original histogram rule is redundant, since this test is implicitly included in the test that checks for a value appearing more than 256 times in a block. The same reasoning applies to our modified histogram rule. Our modified histogram rule triggers if there are 512 or more 2-byte values in a block. This single test ensures that a block will be flagged if a 2-byte value accounts for more than one-fourth of a block or if there are only four or less distinct 2-byte values.

To check how many blocks in our Govdocs database would be flagged by our new rule, we ran it against the set of blocks in the Govdocs database. We found that less than 5% of files in the Govdocs corpus have over 90% of blocks flagged by our new rule, which we found to be an acceptable degree of loss.

5.7 Experiment 4: Sector Matching with a Modified Rule-Based Non-Probative Block Filter

For this experiment, we classify a positive as a target file that has any block match a sector on a drive, as long as the block has not been flagged as non-probative by the modified histogram rule. We classify a negative as a target file where no sectors match a block in the target file, or the only sectors that match are flagged by the modified histogram rule.

We ran our modified histogram rule against our set of RDC block matches to the Govdocs database. Our modified histogram rule flagged 32,172 out of 202,344 unique block matches, or about 16%, compared to 14% with the 3 *ad hoc* rules. In addition, our modified histogram rule flagged 6,669,438 out of 7,819,881 total matches, about 85.3%, which was slightly more than the 85.1% of total matches flagged by the *ad hoc* rules. The modified histogram rule also flagged the 283 rgistdPara array blocks and 5-byte n-gram block described in Section 5.5.2. All of the blocks flagged by the original 3 *ad hoc* rules were included in the set of blocks flagged by the modified hist rule.

We repeated the procedure described in Section 4.2.1 with the 170,172 remaining blocks not flagged by our rule and were able to match 20,604 out of the 170,172 blocks, or about 12%, to 116 true positive files. Since we are assuming that all of these matches are probative, this means that 88% of the remaining blocks were non-probative, similar to the original *ad hoc* rules. However, we found that the modified histogram rule had a higher accuracy and a lower false positive rate than the three *ad hoc* rules. Table 5.10 shows a confusion matrix for the modified histogram rule.

	Actual Positive	Actual Negative
Predicted Positive	116	7,778
Predicted Negative	0	907,152

Table 5.10: Confusion matrix for the modified histogram rule non-probative block filter experiment. The accuracy for the modified histogram rule was 99.15%, while the true and false positive rates were, 100% and 0.85%, respectively.

5.8 Phase II: Hash-Based Carving of JPEGs

In this section we discuss using entropy thresholds to identify Govdocs JPEG files present on RDC drives. First, we discuss the set of Govdocs JPEG files present on RDC drives that we are using as known true matches (i.e., our ground truth). Next, we discuss the results of using different entropy thresholds to identify block matches to known true files. Finally, we present the results of our final entropy threshold after using it on the entire set of 1,530 RDC drives.

5.8.1 Finding a Set of Govdocs JPEGs Present on RDC Drives

After narrowing our test set to just JPEG files, we define ground truth as follows: a target file is considered present if the drive contains over 90% of the target file's blocks in sequence, or if all but one block of the target file matches to the drive, as long as the target file is larger than 2 blocks. We included this second check because we observed Govdocs JPEG files that were actually present on a drive, but that had less than a 90% match, (e.g., 4 out of 5 blocks matched). We exclude JPEG files smaller than 2 blocks for this test because matching 1 block out of 2 only gives a 50% match. In our analysis, we found that these blocks were JPEG header blocks that were not probative for the Govdocs file they matched to, resulting in a false match. We also exclude blocks with an entropy value of 11, since we found that these high entropy blocks come from JPEG Quantization Tables, which are commonly found in JPEG files [2]. After applying these conditions to our set of matches, we identified 34 potential Govdocs JPEGs that we believed were present on RDC drives.

We selected the two RDC drives with the highest number of potential true positive matches to Govdocs JPEG files, AT001-0039 from Austria, and TR1001-0001 from Turkey. AT001-

0039 had 8 potential true positive matches, while TR1001-0001 had 9. We were able to manually extract all 17 of these matches from their respective drives and confirmed that they were true positive matches to JPEG files in the Govdocs Corpus.

For experiments 5-7, we classify a positive as a target JPEG file that has any block match a sector on a drive, as long as the block has not been flagged as non-probative by an entropy threshold of 10.9. We classify a negative as a JPEG target file where no sectors match a block in the target file, or the only sectors that match are below an entropy threshold of 10.9.

5.9 Experiment 5: Sector Matching with an Entropy-Based Non-Probative Block Filter on AT001-0039

With no entropy threshold in place, bulk_extractor reports matches to 76 unique JPEG files in the Govdocs corpus for AT001-0039. Of these matches, 8 are actually present, while 68 are not present. With an entropy threshold of 10.9, we were able to remove all 68 false positive matches and keep all but one of the true positive matches, resulting in one false negative. Table 5.11 shows a confusion matrix for an entropy threshold of 10.9.

Figure 5.13 shows the results of using different entropy thresholds on AT001-0039. The graph on the left shows the Precision vs. False Positive Rate (FPR) for entropy values ranging from 0–10.9. A block is excluded by an entropy threshold if its entropy value is below the threshold value. The graph on the right shows Recall (True Positive Rate) vs. Precision for the same range of entropy values. With an entropy threshold of 10.9, we achieved a precision of 100%, a false positive rate of 0%, and a Recall of 87.5%. The accuracy for the entropy threshold of 10.9 was 99%.

	Actual Positive	Actual Negative
Predicted Positive	7	0
Predicted Negative	1	68

Table 5.11: Confusion matrix for using an entropy threshold of 10.9 as a non-probative block filter on AT001-0039. The accuracy for an entropy threshold of 10.9 was 99%, while the true and false positive rates were, 87.5% and 0%, respectively.

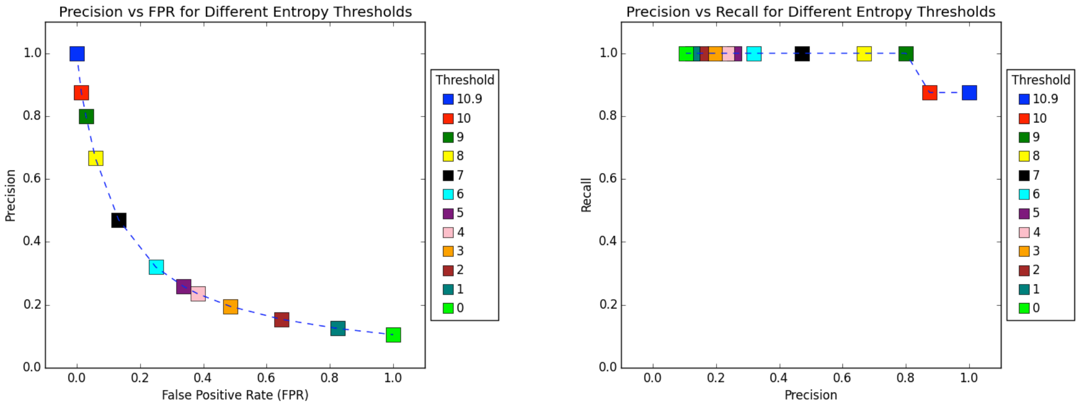


Figure 5.13: This figure shows the results of using entropy thresholds to find Govdocs JPEG files on RDC drive AT001-0039. The graph on the left shows how precision and the false positive rate vary with entropy thresholds ranging from 0–10.9. The graph on the right shows how precision and the recall rate vary with the different entropy thresholds. A block with an entropy value below the entropy threshold is discarded as non-probative. As the entropy threshold increases, so does the precision. At an entropy threshold of 10.9, precision reaches 100%, while the false positive rate falls to 0%. However, the graph on the right shows the recall rate fall to 87.5% once the threshold level reaches 10.

5.10 Experiment 6: Sector Matching with an Entropy-Based Non-Probative Block Filter on TR1001-0001

With no entropy threshold in place, bulk_extractor reports matches to 129 unique JPEG files in the Govdocs corpus for TR1001-0001. Of these matches, 9 are actually present, while 120 are not present. With an entropy threshold of 10.9, we were able to remove all 120 false positive matches and were able to keep all true positive matches. Table 5.12 shows a confusion matrix for an entropy threshold of 10.9.

Figure 5.14 shows the results for TR1001-0001. The graph on the left shows the Precision vs. False Positive Rate (FPR) for entropy values ranging from 0–10.9. The graph on the right shows Recall (True Positive Rate) vs. Precision for the same range of entropy values. With an entropy threshold of 10.9, we achieved a precision of 100%, a false positive rate of 0%, and a recall of 100%. The accuracy for the entropy threshold of 10.9 was 100%.

	Actual Positive	Actual Negative
Predicted Positive	9	0
Predicted Negative	0	120

Table 5.12: Confusion matrix for using an entropy threshold of 10.9 as a non-probative block filter on TR1001-0001. The accuracy for an entropy threshold of 10.9 was 100%, while the true and false positive rates were, 100% and 0%, respectively.

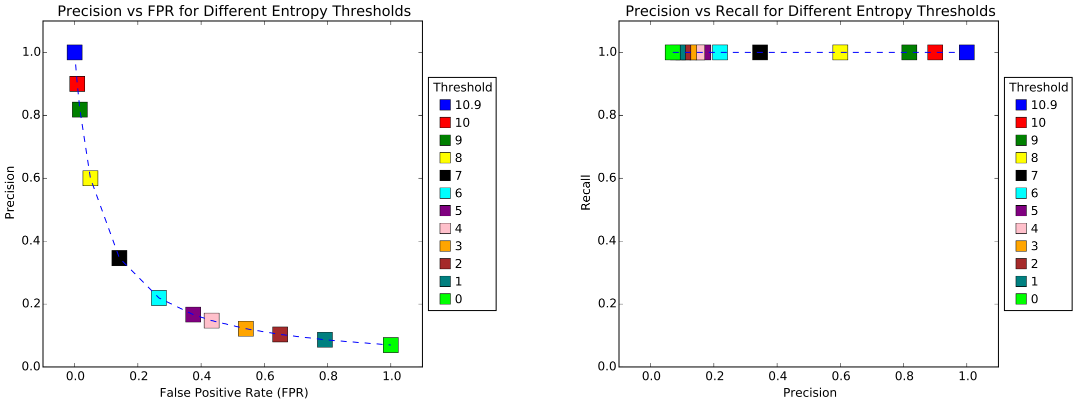


Figure 5.14: This figure shows the results of using entropy thresholds to find Govdocs JPEG files on RDC drive TR1001-0001. The graph on the left shows how precision and the false positive rate vary with entropy thresholds ranging from 0–10.9. The graph on the right shows how precision and the recall rate vary with the different entropy thresholds. A block with an entropy value below the entropy threshold is discarded as non-probative. As the entropy threshold increases, so does the precision. At an entropy threshold of 10.9, precision reaches 100%, while the false positive rate falls to 0%. The graph on the right shows that the recall rate remains at 100% with an entropy threshold of 10.9.

5.11 Experiment 7: Sector Matching with an Entropy-Based Non-Probative Block Filter on 1,530 Drives in the RDC

With no entropy threshold in place, bulk_extractor reports matches to 1,107 unique JPEG files in the Govdocs corpus for all 1,530 RDC drives. Out of the 1,107 matches, 34 are actually present, while 1,073 are not present. An entropy threshold of 10.9 returned matches to 40 unique JPEG files. Table 5.13 shows a confusion matrix for an entropy threshold of 10.9. With our rule, we achieved a precision of 80%, a false positive rate of 0.7%, and a recall of 94%. The accuracy for the entropy threshold of 10.9 was 99%.

	Actual Positive	Actual Negative
Predicted Positive	32	8
Predicted Negative	2	1065

Table 5.13: Confusion matrix for using an entropy threshold of 10.9 as a non-probative block filter on 1,530 RDC drives. The accuracy for an entropy threshold of 10.9 was 99%, while the true and false positive rates were, 94% and 0.7%, respectively.

5.12 Results Summary

Table 5.14 shows a summary of all confusion matrices in Phase I experiments involving general hash-based carving of all file types. Table 5.15 shows a summary of all confusion matrices in Phase II experiments that only involved hash-based carving of JPEGs.

Exp 1	Actual Positive	Actual Negative
Predicted Positive	116	18,731
Predicted Negative	0	896,199
Exp 2	Actual Positive	Actual Negative
Predicted Positive	116	8,985
Predicted Negative	0	905,945
Exp 3	Actual Positive	Actual Negative
Predicted Positive	116	10,797
Predicted Negative	0	904,133
Exp 4	Actual Positive	Actual Negative
Predicted Positive	116	7,778
Predicted Negative	0	907,152

Table 5.14: Confusion matrices for all Phase I experiments involving general hash-based carving of all file types. Exp 1: Naïve sector matching; Exp 2: Filtering non-probative blocks with the *ad hoc* rules; Exp 3: Filtering non-probative blocks with an entropy threshold of 6; Exp 4: Filtering non-probative blocks with a modified histogram rule.

Exp 5	Actual Positive	Actual Negative
Predicted Positive	7	0
Predicted Negative	1	68

Exp 6	Actual Positive	Actual Negative
Predicted Positive	9	0
Predicted Negative	0	120

Exp 7	Actual Positive	Actual Negative
Predicted Positive	32	8
Predicted Negative	2	1065

Table 5.15: Confusion matrices for all Phase II experiments involving hash-based carving of JPEGs only. Exp 5: Filtering non-probative blocks on AT001-0039 with an entropy threshold of 10.9; Exp 6: Filtering non-probative blocks on TR1001-0001 with an entropy threshold of 10.9; Exp 7: Filtering non-probative blocks on 1,530 RDC drives with an entropy threshold of 10.9.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6: Conclusion

6.1 Summary

Digital forensic investigators need an alternative method of finding target files on digital storage media. Traditional hash-based file identification techniques that rely on exact hash matches to target files in a database are susceptible to the avalanche effect, where a small change to a target file completely changes the value of its hash. Target files on searched media that have been intentionally modified, partially deleted, or corrupted will not match known files in a database. Valuable evidence may be missed this way. In addition, traditional file identification techniques rely on the file system to find target files, which is time consuming and error prone.

In order to solve these problems Garfinkel [2] proposed using sector hashes to find target files on searched media. Sector hashing solves many of the problems associated with traditional file hashing techniques. First, it does not require an entire target file to be present on a drive for matches to be returned. With enough sector hash matches to a target file, a forensic investigator may be able to determine its presence on a drive. Second, sectors can be read and hashed sequentially off of searched media without relying on filesystem metadata, such as disk partitions. This allows forensic investigators to break up a drive into chunks and analyze them in parallel. Since sector hashing is nearly file system agnostic, it is faster than conventional methods that require forensic investigators to process a drive's contents by seeking back and forth across the drive following file system pointers. Finally, a third benefit of sector hashing are probative blocks. Probative blocks are blocks that help forensic investigators demonstrate that a target file resides on a drive. A single probative block can be used to demonstrate that a file is or was once present on a drive.

However, sector hashing presents its own set of problems. Not all blocks are probative blocks, or blocks that can be used to demonstrate the presence of a target file with high probability. Many blocks are programmatically generated and are found in many files. Sector hash matches to these blocks are false positives for a target file, since they do not match content generated by a user, but instead match data structures common across files.

Garfinkel and McCarrin [4] proposed using three rules to identify common blocks: the histogram, whitespace, and ramp rules. They called these 3 rules the *ad hoc* rules. We evaluated the *ad hoc* rules by using the million file Govdocs corpus to create a database of target block hashes and by using 1,530 drives from the Real Data Corpus (RDC) as our set of searched media. We found that the *ad hoc* rules flagged 85% of the total sector hash matches from RDC drives to block hashes in the Govdocs database. An entropy threshold of 7 flagged an equivalent number of common blocks, but also flagged over 90% of blocks for 12.5% of files in the Govdocs corpus, compared to 1.65% for the *ad hoc* rules. When over 90% of blocks in a file are flagged, the probability of carving the file is greatly reduced [4]. In order to not suppress a large number of probative blocks, we tested the *ad hoc* rules against an entropy threshold of 6, which only flagged over 90% of blocks for less than 5% of files in the Govdocs corpus. We found that filtering non-probative blocks with the *ad hoc* rules resulted in a true positive rate of 100%, a false positive rate of 0.98%, and an accuracy of 99.02%. Filtering non-probative blocks with an entropy threshold of 6 resulted in a true positive rate of 100%, a false positive rate of 1.18%, and an accuracy of 98.82%. Since the false positive rate was lower for the *ad hoc* rules and the accuracy was higher for the *ad hoc* rules, we preferred the *ad hoc* rules over a simple entropy threshold of 6 when searching for target files in general.

There were several limitations to the *ad hoc* rules, however. First, since the histogram and ramp rules read a block as 1024 4-byte values, they failed to detect many patterns, such as the 18-byte rgistdPara array Microsoft Word structure described in Section 5.5.2. Second, they failed to flag font structures present in PDF files, resulting in a large number of false positive matches for that file type. Third, the *ad hoc* rules only flagged about 14% of unique block matches returned by scanning the RDC drives, which resulted in a precision of only 1.27% due to the number of false positive matches.

We found that a modified histogram rule that read blocks as 2048 2-byte values instead of 1024 4-byte values flagged the patterns that the original histogram rule failed to detect, including the rgistdPara array mentioned above. In addition, our modified histogram rule captured all of the blocks flagged as non-probative by the three *ad hoc* rules. Filtering non-probative blocks with the modified histogram rule resulted in a true positive rate of

100%, a false positive rate of 0.85%, an accuracy of 99.15%, and a precision of 1.47%. Since the modified histogram rule outperforms the three *ad hoc* rules on all measures, we recommend that the *ad hoc* rules be replaced by the single modified histogram rule.

When searching for specific file types, we found that entropy thresholds can be used to identify probative blocks. After focusing our search efforts on JPEG files, we found that an entropy threshold of 10.9 was able to identify Govdocs JPEG files on RDC drives with 80% precision and 99% accuracy. We believed this occurred because JPEG blocks of user-generated content have high entropy values. Thus, we were able to identify probative blocks by filtering blocks with an entropy value below our threshold.

6.2 Future Work and Lessons Learned

There is still significant room for improvement when it comes to identifying non-probative blocks. While we believe that we have dealt with the majority of common blocks that contain easy to recognize patterns, such as repeating n-grams or ramp structures, it is difficult to imagine rules for blocks that do not follow any obvious pattern, such as fonts. One approach for dealing with these blocks is to create a whitelist of known common blocks. If a block is encountered that matches the whitelist, it is eliminated from being considered a probative block. For example, we observed that the top 50 sector matches to blocks in our Govdocs database accounted for over half (50.88%) of the total matches returned. These blocks could be added to a whitelist to reduce the number of false positive matches, since we know that they are non-probative blocks consisting of repeating n-grams and pieces of font structures.

Another approach would be to use a byte shifting algorithm to detect similar patterns within blocks that occur at different alignments. For example, we observed that 10 out of the top 50 matches to blocks in our Govdocs database consisted of the same 131-byte n-gram aligned at different start offsets.

While a high entropy threshold could be used to flag non-probative blocks, one needs to understand the entropy distribution of specific file types in order to avoid suppressing a large percentage of probative blocks. We successfully applied this approach to JPEG files and believe that entropy thresholds can be used to identify other file types with high precision and accuracy.

After comparing the results from our hash-based carving experiments involving all file types with the results from our JPEG only experiments, it is clear that general rules do not work as well as file specific rules. Thus, we recommend that hash carving approaches be tailored to suit the file type being searched.

References

[1] W. Stallings and L. Brown, *Computer Security: Principles and Practice*, 2nd ed. Prentice Hall, 2011.

[2] S. Garfinkel *et al.*, “Using purpose-built functions and block hashes to enable small block and sub-file forensics,” *Digital Investigation*, vol. 7, pp. S13–S23, 2010.

[3] J. Taguchi, “Optimal sector sampling for drive triage,” master’s thesis, Naval Postgraduate School, Monterey, CA, June 2013.

[4] S. Garfinkel and M. McCarrin, “Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb,” submitted to DFRWS 2015.

[5] J. E. Silva, “An overview of cryptographic hash functions and their uses,” January 2003, cited on February 10, 2015. [Online]. Available: http://www.sans.org/reading-room/whitepapers/vpns/overview-cryptographic-hash-functions-879

[6] E. Thompson, “Md5 collisions and the impact on computer forensics,” *Digital Investigation*, vol. 2, no. 1, pp. 36–40, 2005.

[7] F. Breitinger and H. Baier, “Performance issues about context-triggered piecewise hashing,” in *Digital forensics and cyber crime*. Springer, 2012, pp. 141–155.

[8] National Institute of Standards and Technology. (2015, February). National Software Reference Library Reference Data Set. [Online]. Available: http://www.nsrl.nist.gov.

[9] Federal Bureau of Investigation. (2003, May). Privacy impact assessment Child Victim Identification Program innocent names national initiative. [Online]. Available: http://www.fbi.gov/foia/privacy-impact-assessments/cvip.

[10] V. Roussev, “Data fingerprinting with similarity digests,” in *Advances in digital forensics vi*. Springer, 2010, pp. 207–226.

[11] S. L. Garfinkel, “Digital forensics research: The next 10 years,” *digital investigation*, vol. 7, pp. S64–S73, 2010.

[12] J. Young *et al.*, “Distinct sector hashes for target file detection,” *Computer*, vol. 45, no. 12, pp. 28–35, 2012.

[13] J. Kornblum, “Identifying almost identical files using context triggered piecewise hashing,” *Digital Investigation*, vol. 3, pp. 91–97, 2006.

[14] J. Kornblum. (2014, October). SSDeep version 2.12. [Online]. Available: http://ssdeep.sourceforge.net.

[15] V. Roussev. (2013, October). SDHash version 3.4. [Online]. Available: http://roussev.net/sdhash/sdhash.html.

[16] K. Foster, "Using distinct sectors in media sampling and full media analysis to detect presence of documents from a corpus," master's thesis, Naval Postgraduate School, Monterey, CA, September 2009.

[17] S. Garfinkel *et al.*, "Bringing science to digital forensics with standardized forensic corpora," *Digital Investigation*, vol. 6, pp. S2–S11, 2009.

[18] J. Kornblum. (2014, May). md5deep version 4.4. [Online]. Available: http://md5deep.sourceforge.net.

[19] Forensics Wiki (2013, November). Category:Digital Forensics XML. [Online]. Available: http://www.forensicswiki.org/wiki/Category:Digital_Forensics_XML.

[20] B. Allen. (2015, January). hashdb version 1.0.0. [Online]. Available: https://github.com/simsong/hashdb/.

[21] United States Navy (2014, September). NPS High Performance Computing. [Online]. Available: https://wiki.nps.edu/pages/viewpage.action?title=Home&spaceKey=HPC.

[22] Forensics Wiki (2015, January). Encase image file format. [Online]. Available: http://www.forensicswiki.org/wiki/Encase_image_file_format.

[23] Forensics Wiki (2014, December). Libewf. [Online]. Available: http://www.forensicswiki.org/wiki/Libewf.

[24] Forensics Wiki (2014, June). Raw Image Format. [Online]. Available: http://www.forensicswiki.org/wiki/Raw_Image_Format.

[25] B. Carrier (2011, November). tsk_loaddb. [Online]. Available: http://www.sleuthkit.org/sleuthkit/man/tsk_loaddb.html.

[26] B. Carrier (2010, March). Mmls. [Online]. Available: http://www.sleuthkit.org/sleuthkit/man/mmls.html.

[27] B. Carrier (2008, September). Icat. [Online]. Available: http://www.sleuthkit.org/sleuthkit/man/icat.html.

[28] SleuthkitWiki (2010, January). Blkls. [Online]. Available: http://www.sleuthkit.org/sleuthkit/man/blkls.html.

[29] J. Weigert., (1996, August). xxd(1). [Online]. Available: http://linuxcommand.org/man_pages/xxd1.html.

[30] FreeBSD (2014, April). DD(1). [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=dd(1)&sektion=.

[31] Microsoft (2009, July). Microsoft Office Visualization Tool (OffVis) version 1.0. [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=2096.

[32] AWare Systems (n.d). AsTiffTagViewer version 2.00. [Online]. Available: http://www.awaresystems.be/imaging/tiff/astifftagviewer.html.

[33] Forensics Wiki (2014, June). Bulk extractor. [Online]. Available: http://www.forensicswiki.org/wiki/Bulk_extractor.

[34] SleuthkitWiki (2014, January). TSK Tool Overview. [Online]. Available: http://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview.

[35] I. Kaplan (2002, August). Shannon Entropy. [Online]. Available: http://www.bearcave.com/misl/misl_tech/wavelets/compression/shannon.html.

[36] AWare Systems (n.d). TIFF Tag ColorMap. [Online]. Available: http://www.awaresystems.be/imaging/tiff/tifftags/colormap.html.

[37] Adobe (2005, September). XMP Specification. [Online]. Available: http://partners.adobe.com/public/developer/en/xmp/sdk/XMPspecification.pdf.

[38] International Color Consortium (n.d). Introduction to the ICC profile format. [Online]. Available: http://www.color.org/iccprofile.xalter.

[39] Microsoft (2015). 3.7 Example of a List. [Online]. Available: https://msdn.microsoft.com/en-us/library/dd945358(v=office.12).aspx.

[40] Microsoft (2015). 2.9.147 LSTF. [Online]. Available: https://msdn.microsoft.com/en-us/library/dd907589(v=office.12).aspx.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California